COMP5310 - Assignment 2

sbir0140 (460374219) kmag0848 (540275221) wcui0671 (530414443)

May 7, 2024

1 Setup

1.1 Topic and research question

With climate change becoming an ever increasing concern, and the effects of wildfires and droughts devastating Australia, it is important to grasp a better understanding of Australian weather patterns and identify repeating behaviours (if there are any) so that we may predict and prepare for future natural disasters. Weather monitoring has been an on-going process and interest of the Australian government for a long time, having established over 17933 weather stations across Australia since 1826 [1]. Weather predictions benefit many stakeholders including citizens, farmers, agricultural businesses, investors and more, allowing us to prepare for rain, heat, strong wind, snow, and hail. By recording and analysing weather data, we have been able to remove some of the randomness of weather and improve our quality of life in general.

The research question that we explore in this paper is identifying any rainfall patterns that may exist and what the correlated weather metrics are that influence this rainfall, and if these metrics can be used to predict if rainfall may occur on the current day or the next. More concisely, are there any correlated weather patterns that can be used to determine when rainfall will occur. By answering this research question, stakeholders will be better prepared for wet weather conditions.

1.2 Dataset

The dataset was acquired from Kaggle [2], and contains daily recorded rainfall data from numerous weather stations across Australia. The data from the dataset was retrieved from the Bureau of Meteorology over a period of 10 years from 2007 to 2017. It has been thoroughly cleaned and processed, removing outliers and missing values, preparing it for model training. There are 23 attributes and $\tilde{1}$26k instances in this dataset. The data dictionary of the attributes can be found in appendix 6.

The primary challenges faces in utilising this dataset for training, was identifying the most suitable features for predicting the `RainTomorrow` column. There is a even mix of numerical and categorical features, however it is difficult to utilise categorical data in the models we have chosen, even resulting in poorer performance. These features often have a low or no correlation to our prediction column and so experimentation and careful feature selection was required to reduce our feature size.

1.3 Modelling agreements

The measures of success that we decided to use for our performance evaluation include using a Confusion Matrix, an AUC-ROC curve and a F1-Score. The use of the Confusion Matrix provides a general overview of the distribution of successful and failed classifications. This is helpful in assessing the impact of the model if it were predicting high False Negatives, which in our case would have a higher negative impact than False Positives, since we are predicting it won't rain, but it does. The AUC-ROC curve represents this information in another format, allowing us to see how well the model performs in distinguishing between classes. Both the AUC-ROC curve and F1-Score are similar, but provide a different view of the same metric. The F1-Score is a numerical statistic we can use to tune our models and compare our models against. With this statistic, we can create loss functions the attempt to maximise this statistic.

2 Modeling: individual component (530414443)

2.1 Predictive model: neural network

2.1.1 Model description

An artificial neural network (ANN) is a computational model inspired by the structure and function of biological neural networks in the human brain. It consists of interconnected nodes, or neurons, organized into layers: an input layer, one or more hidden layers, and an output layer. ANNs can be used for various tasks, including classification, regression, pattern recognition, and reinforcement learning. They have gained popularity due to their ability to model complex, nonlinear relationships in data and their success in solving a wide range of real-world problems in fields such as image and speech recognition, natural language processing, and autonomous vehicles.

As for the assumption needed for applying ANN to the dataset, we will illustrate from following perspective: From the perspective of data availability: the dataset have a sufficient amount of data samples(140787*19) comprising corresponding input features and target outputs which is required by ANN model. From the perspective of data quality: After data cleaning, the dataset have been accurate, complete and clean. Also, the distribution of each column is same after checking because ANNs typically assume that the data is independently and identically distributed.

In the previous stage, we used a correlation matrix to observe the correlation between features in the dataset and found that most features did not exhibit strong correlations. In response to this phenomenon, neural networks perform well in handling datasets with uncorrelated features. This dataset originates from real data collection, thus may contain noise and incomplete information. However, ANNs can effectively handle noisy and incomplete datasets. Moreover, the automatic learning capability of ANNs allows them to automatically learn relevant features from raw data without manual feature engineering, reducing dependence on feature engineering and making model development more efficient and flexible. Despite the notable advantages of ANNs, we still need to address and mitigate the drawbacks associated with training datasets using neural networks. For example, training ANNs requires large amounts of data and computational resources, as well as careful tuning of hyperparameters. ANNs are susceptible to overfitting, especially when the model is overly complex, leading to memorization of the training data rather than learning generalizable patterns. Therefore, in the design process of neural networks, finding the appropriate model complexity is crucial.

Based on the exploratory data analysis and comprehensive analysis of the dataset's characteristics, we believe that neural networks are highly suitable for the classification prediction task of this dataset.

2.1.2 Model algorithm

The main body of a neural network model is made up of several parts. First of all, the front-end is the input layer, which directly receives the original data, and its adaptability determines the completeness of the original data information reception. Next, the raw data is fed layer by layer through weighting and activation functions in each hidden layer until the final output layer produces the final prediction. On top of the neural network's infrastructure, it also has back propagation: an algorithm for training neural networks that iteratively adjusts the network's parameters based on the computational gradient of the loss function relative to these parameters

In the realm of artificial neural networks, a multitude of parameters stand pivotal in shaping the model's behavior and performance. At the forefront lies the architecture, encompassing the number of layers, the size of each layer, and the connectivity patterns between neurons. The choice of activation functions, such as sigmoid, ReLU, or tanh, profoundly influences the network's ability to capture nonlinear relationships within the data. Additionally, the selection of optimization algorithms and their associated hyperparameters, such as learning rate and momentum, significantly impacts the training dynamics and convergence speed. Regularization techniques, including dropout and L1/L2 regularization, play a crucial role in mitigating overfitting by controlling the model's complexity. Furthermore, initialization methods for weight parameters and batch normalization parameters contribute to the stability and efficiency of the learning process. Overall, fine-tuning these parameters requires a nuanced understanding of their interplay, and their judicious adjustment can unlock the full potential of neural network models.(pseudo code are in the end)

I chose a neural network model with two layers as the prediction initial model, which is an appropriate model com-

plexity, which is very important for the performance and generalization ability of the model. It can adjust underfitting and overfitting to find an equilibrium point. At the same time, the model can avoid being too simple to capture the complex relationships in the data, and can also avoid overfitting due to being too complex, which will reduce the performance of the model. At the same time, the computing cost and training time can also be controlled within a relatively ideal range

2.1.3 Model development

One of the advantages of neural networks is that compared with traditional machine learning algorithms, they have a weaker demand for feature engineering. Through end-to-end learning, neural networks can directly learn the mapping relationship between feature representations and models from raw data, reducing the dependence on manual feature engineering. So, on top of the cleaned data, we directly separate the feature and prediction labels. Since the neural network in the PyTorch library requires input data in the form of tensors, we use **values.astype(float)** to ensure that the format of the input features is uniformly converted to float, and **values.astype(int)** converts the label column to the integer format. Finally, use **torch.Tensor( )** converts all data into the form of tensors.

Next, the training set and test set are divided. According to the principle of hierarchical sampling, and based on the previous test of the balance of the dataset, the sample classification ratio is about 2:8, so we use the **train-test-split( )** function to divide the features and labels into the training set and the test set according to the ratio of 20% test set to 80% training set.'**batchsize = 64**' sets the number of samples per batch to 64. '**shuffle=True**' shuffles the order of the data before each epoch to increase the robustness of the model.

Due to the property of ANN that include many modules and parameters to tun, we want to figure out the best combination of these parameters. So we use **grid search** to find out model performance on each parameter combination. Learning rates = [0.001, 0.01, 0.1], weight decays = [0.0001, 0.001, 0.01], dropout rates = [0.2, 0.35, 0.5].

In model design, we first create a class named ANNnet using class '**ANNnet(nn.Module)**'. Next, we call the constructor of nn.Module using '**super().init()**'. Moving into the core of the neural network, we define the input layer 'self.input', which maps the number of features to 64 nodes. '**self.fc1**' and '**self.fc2**'define two fully connected layers, with the first mapping from 64 nodes to 128 nodes, and the second from 128 nodes to 256 nodes. 'self.output' defines the output layer, which maps the 256 nodes from the last hidden layer to the number of output categories.

In the forward propagation function 'forward', **F.leaky.relu( )** applies the Leaky ReLU activation function to the input layer. **F.dropout(x, .5)** applies 50% dropout to prevent overfitting, randomly zeroing input elements during training to prevent the model from overly relying on any single input, thus enhancing model generalization. This process is repeated on the two hidden layers, using Leaky ReLU activation and dropout. Finally, the result of the output layer is directly returned for loss calculation.During backpropagation, we define the loss function by **nn.CrossEntropyLoss( )**, which is the cross-entropy loss function commonly used for multi-class classification problems where model outputs need to be transformed into probability distributions through the softmax function. Then, we define the optimizer by **torch.optim.Adam( )** using the Adam optimizer to adjust network weights. Setting '**lr=.001**' establishes a learning rate of 0.001, a parameter controlling the model's learning speed. A higher learning rate adjusts weights faster, but excessively high rates may lead to training instability.

2.2 Model evaluation and optimization

2.2.1 Model evaluation

First, we get the confusion matrix, and the results are as follows

	YES TRUE	NO TRUE
YES pred	1045513	49537
NO pred	182276	130574

Table 1: Confusion Matrix

In the output, we get an accuracy of up to 83.68%, which seems to be a relatively desirable value, but due to the imbalance of the dataset, the accuracy is not entirely informative.

Therefore, the model performance needs to be evaluated in conjunction with the loss function, and in this training, the loss function is stable at about 0.39, which confirms that the accuracy results show effective learning and generalization.

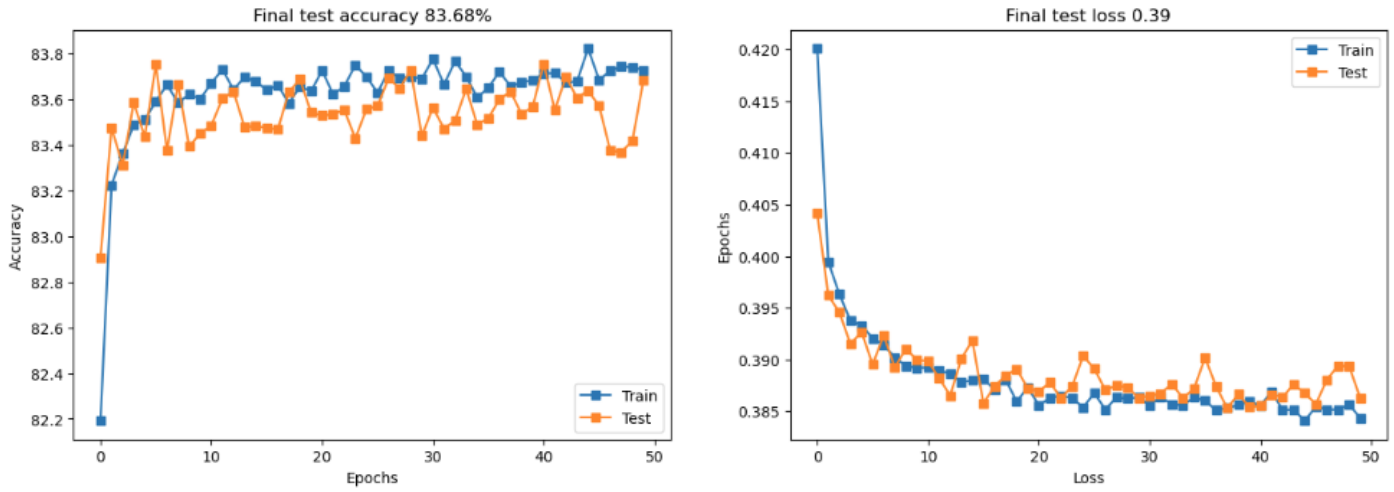

Figure 1: Accuracy and Loss

Computationally, we get an F1 score of 0.83, which is generally a good reference metric for an unbalanced dataset because it aims to find the balance between precision and recall. Using the ROC curve, we found that the area under the ROC curve was 0.69.It's better than a random guess (that will be an AUC for 0.5) but still show quite space for improvement.

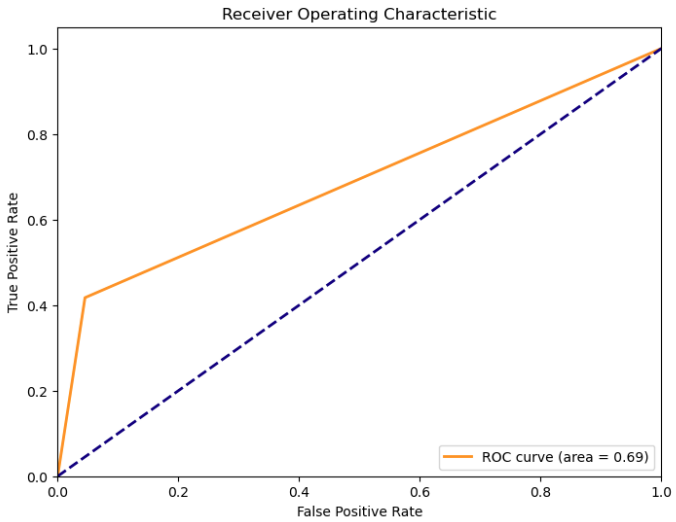

Figure 2: AUC-ROC curve

2.2.2 Model optimization

Neural networks have so many parameters that can be tuned that the best combination of parameters needs to be figured out to get the best performing model. At the same time, it can also improve the efficiency of parameter combination search. In this experiment, we perform a grid search for all parameter combinations of learning rate, weight decay, and dropout rate, and output the group with the highest accuracy. The following results were obtained: learning rate=0.1, weight decay=0.01, dropout rate=0.5.

3 Modeling: individual component (460374219)

3.1 Predictive model: Logistic Regression

3.1.1 Model description

Logistic Regression is a supervised machine learning model that is used for classification. There are three types of logistic regression models including Binary, Multinomial and Ordinal. For this task, I am using a Binary classifier. Logistic regression assumes a linear relationship between features, and that outliers have been removed. In addition, it is important to consider multicollinearity, since this can influence the accuracy of coefficient estimation and may lead to ineffective model training if using regularisation. Calculating the Variation Inflation Factor (VIF) for each feature is a good technique to identify multicollinearity.

The features used for this model were selected using both the Chi-Squared test and VIF, choosing the 4 best features. We do see a linear relationship between some of the features that result from this process, but not all are, so we already identify that this model may have some limitations on the accuracy it can achieve.

Another assumption of Logistic Regression models is that the features are assumed to be normalised. This requires features to be scaled appropriately so that each feature is comparable. This can be done using the `StandardScalar` class from `sklearn`. Secondly, it is important to consider over-fitting, and so using `L1` or `L2` regularisation techniques is an important technique in the training process. Another approach to work around non-linear relationships is using polynomial features. This may help improve the performance of the model.

3.1.2 Model algorithm

Logistic regression is based on the linear model $f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w}\mathbf{x}^{(i)} + b$, where $\mathbf{w}, \mathbf{x}^{(i)}$ are vectors. What makes logistic regression a binary classifier is the threshold set when generating the prediction. In a linear model, the output z scales from $(-\infty, \infty)$, and when parsed through the sigmoid function, the output ranges from $[0, 1]$. The formula for the sigmoid function is shown below:

$$g(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{z}}} = \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x}^{(i)} + \mathbf{b})}}$$

Here, $\mathbf{x}^{(i)}$ is this equation represents a single sample.

The threshold condition is represented as:

$$pred = \begin{cases} 0 & \text{if } z < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

In a linear model, the loss function used is the Square Error cost. however in Logistic Regression the loss function is conditionally calculated based on the predicted class, and is represented as a decreasing slope on a graph. The Logistic loss function is represented by the following condition:

$$loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) = \begin{cases} -\log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$
$$= -y^{(i)} \log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right)$$

The reason the loss function is not the Squared Error cost is because it creates many local minima that the model can fall into, therefore not achieving the best outcome. There are various configurations to the Logistic Regression classifier including specifying the strength regularisation and using a regularisation technique such as L1 (lasso) or L2 (ridge). You can also specify which solver algorithm to use, which is depending on the type of classification being done. Most commonly, the `lbfgs` solver is used. The pseudo-code of this algorithm can be seen in Appendix A.1.

There have been various advancements for the Logistic Regression model including improved regularisation techniques such as Group Lasso and Elastic Net Regularisation, faster solvers such as liblinear and saga, and better feature engineering algorithms. Each of these advancements improve the performance of the Logistic Regression model, whilst also reducing the labour and potential user bias/error in model training. This makes the model more accessible to the general audience.

3.1.3 Model development

We use the model from the `sklearn` library and instantiate it as follows:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
```

Alongside this, we also built a second model using Polynomial features with a degree of 2:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2)
X_test_poly = poly.fit_transform(X_train)
```

The specific degree of dimensionality was determined through manual adjustment.
Our feature selection process utilises the `chi2` method from `sklearn` and we select the 4 best features using `SelectKBest`.

```
from sklearn.feature_selection import chi2, SelectKBest
# Apply SelectKBest class to extract top 4 best features
bestfeatures = SelectKBest(score_func=chi2, k=4)
fit = bestfeatures.fit(X, y)
```

In addition, we calculated the VIF of each feature to determine multicollinearity using the `variance_inflation_factor` method from the `statsmodels.stats.outliers_influence` library. The result gives us the features `["Rainfall", "Humidity9am", "Humidity3pm", "WindGustSpeed"]`, which then proceed to split our data into a test/train set with a 30% split. It is then necessary to perform Standard Scaling and Principle Component Analysis to normalise our features and reduce dimensionality. The benefits of this include increased model performance and convergence speed, as well as feature comparability.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
pipeline = Pipeline([
    ('sc', StandardScaler()),
    ('pca', PCA())
])
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)
```

In the above, we use `sklearn`'s `Pipeline` class to perform a step-by-step transform of the data. We then train the model with `classifier.fit(X_train, y_train)`.

3.2 Model evaluation and optimization

3.2.1 Model evaluation

The performance of our model reached an accuracy of 84.14%. Below can be found the confusion matrix and performance statistics for this model, generated using the `classification_report` from `sklearn.metrics`. From Table 2, we can

	Normal		**Polynomial**	
	Predicted			
Actual	No	Yes	No	Yes
No	28075	1386	28101	1360
Yes	4557	3841	4502	3896
F1-Score	**0.90**	**0.56**	**0.91**	**0.57**

Table 2: Confusion matrix for the predicted and actual values. The Normal and Polynomial models have an accuracy of 84.14 % and 84.32 %, respectively.

see that the difference between normal and polynomial features is negligible and doesn't have a significant impact on our

model's performance. The F1-Score is quite low, and looking at Table 2 we can see that the number of False Negatives are quite high. Our classes for `RainTomorrow` have a ratio of `11:39` for Yes:No, which creates a reasonable skew in the class distribution and may influence the performance results.
The primary concern would be understanding how much of an impact in the real world would a false negative or false positive have. The worst case scenario is of course if it rains and we predicted it wouldn't, as this would have a direct impact on people's daily lives or business activities if they rely on our prediction. In the opposing case, a false positive will simply add an inconvenience of carrying an umbrella or delaying plans, but the impact is much lower. With an AUC

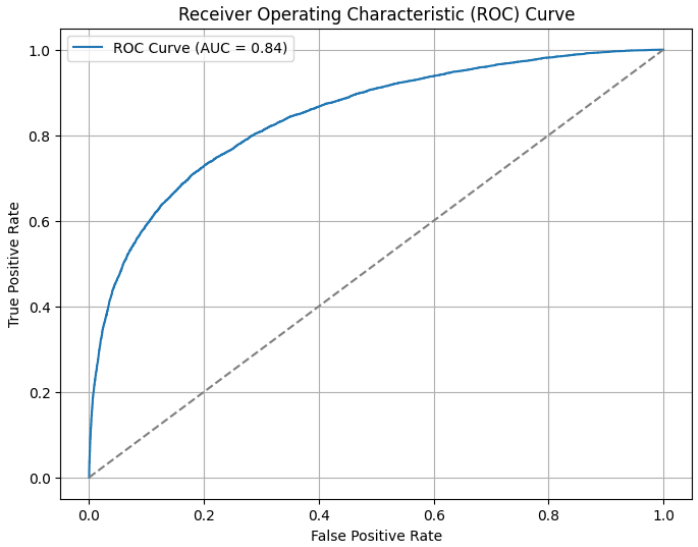

Figure 3: AUC-ROC Curve for the Logistic Regression model

of 0.81, the model has a high chance of distinguishing between positive and negative cases and from the shaped of the curve in Figure 3, we can see that the performance of the model is quite good. However, better performance is required in reducing the number of False Negatives generated by the model. For this, I believe more samples with a positive classification is required.

3.2.2 Model optimisation

For hyper-parameter tuning, we utilised `GridSearchCV` which performs an exhaustive search of every parameter provided using the ranges provided. The specific parameters of our model that we are tuning are `C` and `penalty`. Our value range for `C` is generated with `np.logspace(-4, 4, 50)`, which has a minimum of 0.0001 and maximum of 10,000. For `penalty`, we test the values `["l1", "l2", "elastic", "none"]`. Since there is only a minor difference between polynomial features and normal features, we will only cover the optimisation for normal features. The results returned have `C` = 0.18420699 and `penalty` = `"l2"`. The resulting performance difference however, is negligible if any change at all, with accuracy still resting at 84.14%. From this, we can determine that this is a limitation of the model and the dataset we are using. The threshold

	Predicted	
Actual	No	Yes
No	28075	1386
Yes	4557	3841
F1-Score	**0.90**	**0.56**

Table 3: Confusion matrix for the predicted and actual values. The accuracy of the model is 84.14 %

value for our prediction was experimented with also, comparing threshold values generated by `np.linspace(0.05, 1, 20)`. The best threshold value determined, was 0.5. There are alternative hyper-parameter tuning methods, such as using a train/dev/test split for a more controlled manual fine tuning, however this approach is very time intensive and may not manage to find the best hyper-parameter values. Therefore, using Grid Search has removed a lot of user error from the process whilst improving the efficiency of Hyper-parameter tuning. Both methods are practical and are widely used today.

4 Modeling: individual component (540275221)

4.1 Predictive model: Random Forest

4.1.1 Model description

Model description:

Random Forest algorithm works by creating a number of Decision Trees during Training where each tree is constructed using random subset of data set to measure a random subset of features in each partition with repetition. The algorithm aggregates the results of all trees to make predictions. Random Forest algorithm assumes that the all the features have meaningful values, as missing values or nonsensical values lead to incorrect predictions. The other assumption Random Forest makes is that there should be low correlation between Trees as Random Forest aims to reduce variance by combining predictions from uncorrelated Trees. If the Trees are highly correlated, the ensemble model will not offer better prediction than a single Decision Tree. As the dataset consists of about 140,000 rows and 19 columns which is complex and as we are dealing with high-dimensional dataset, Random Forests are well suited as they can effectively avoid over-fitting

4.1.2 Model algorithm

Random Forest is a powerful ensemble algorithm which works by creating multiple Decision Trees. To understand how Random Forest works, it is essential to know how Decision Trees work. Basically Decision Trees maps observations to a target value by asking a series of questions. This can also be viewed as hierarchy of if else statements. The goal of the Tree is to produce pure leaves (nodes with only one classification) by splitting data based on conditions. The model decides the splitting condition by maximizing the 'Information Gain', which can be quantified using Entropy. Information gain of a node is calculated by:

$$IG(Y|X) = H(Y) - H(Y|X)$$

where Y = class label
X = Attribute
$H(Y)$ = Entropy of Y
$H(Y|X)$ = Entropy of Y given X
Entropy is the measure of the degree of randomness or uncertainty in the data, which can be calculated by :

$$H(Y) = -\sum_{i=1}^{m} p_i log_2(p_i)$$

where m = Number of classes (classifications)
p_i = Probability of the ith class
For a Random Forest, as the name suggests creates multiple decision trees using different random subsets of the data and features. This method of developing multiple weak models on different subsets of the training data with replacement is called Bagging. Each decision tree is like an expert, giving its opinion on how to classify the data and predictions are made by calculating the prediction for each tree, then taking the most popular result (by vote).
A detailed Pseudo code of the algorithm is provided in the appendix.

4.1.3 Model development

For the development of Random Forest model, we used the sklearn's available library

```
from sklearn.ensemble import RandomForestClassifier
```

I checked if there are any missing values in the dataset and dropped all the rows having NaNs as Random Forest Classifier from sklearn does not accept NaN values. To drop rows having NaN

```
data = data.dropna()
```

Next, the 'Date' and 'Location' columns were dropped as they are categorical and does not add any meaning to the final label 'RainTomorrow'. However all the other columns were kept for the model as Random Forest performs better

with multi dimensional data. The data is split vertically into feature set 'features', which contains all features(X) and 'y' containing the result label ('RainTomorrow' column). Both X and y are then converted to arrays. The data is then split horizontally into train data and test data using sklearn's 'train_test_split' module with test_size = 0.3 (30% for test and 70% for train).

```
from sklearn.model_selection import train_test_split
feature_df = data[['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir',
   'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am',
   'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
   'Pressure3pm', 'Temp9am', 'Temp3pm', 'RainToday']]
X = np.asarray(feature_df)
y = np.asarray(data['RainTomorrow'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 643)
```

The main parameters for Random Forest model are:

1) n_estimators : Number of decision trees to be included in the forest
2) max_depth : The maximum depth each decision tree can grow
3) min_samples_split : This determines the number of samples required to split a node in the decision tress
4) min_samples_leaf : Minimum number of samples allowed in a leaf node
5) max_features : Number of features considered at each split in the decision tree. eg 'sqrt', 'log2'

Initially the model is fit with default parameter values by using the function.

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

4.2 Model evaluation and optimization

4.2.1 Model evaluation

Confusion Matrix and F1 score

	Predict	
Actual	0	1
0	28117	1351
1	4161	4230
F1 Score	0.91	0.61

Table 4: Confusion Matrix

With the default parameters, the model gives an accuracy of 85.5%, which is pretty good however based on the confusion matrix we can see that the model seems to be biased towards predicting non-rainy days due high true negative values and relatively low true positive values. Also the model misses a substantial portion of the rainy days (high false negatives).

For class 0, the F1-score is 0.91 which is pretty good indicating good balance between precision and recall and the model is effective in identifying negative cases (i.e., when it will not rain the next day). However for class 1, the F1-score is 0.61 which is comparatively lower than class 0 and suggests a less balanced performance.

4.2.2 AUC - ROC curve

In figure 4, we see that Area under the curve value is 0.88 which signifies that Random Forest model is significantly better than random classification of positives and negatives and has a good ability to differentiate between the classes.

4.2.3 Model optimization

For optimizing the Random Forest Model, RandomizedSearchCV() function from sklearn was used to get the best parameters. RandomizedSearchCV() was used instead of GridSearchCV() as grid search is computationally very expensive for

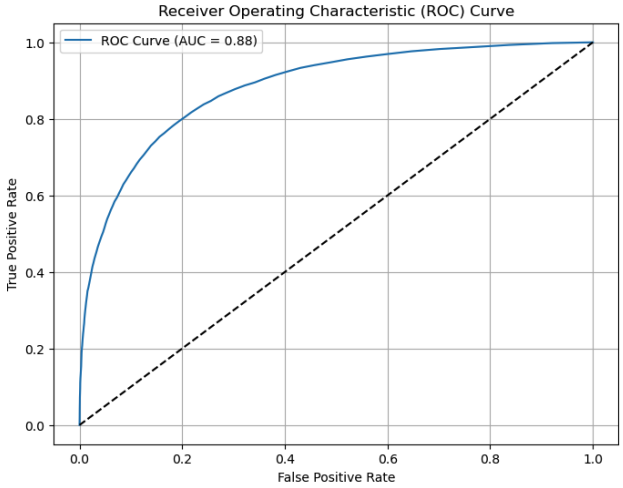

Figure 4: ROC Curve

checking the number of trees.
The following parameters were randomly picked for hyper parameter tuning.
1) n_estimators : From 50 to 500
2) max_depth : from 3 to 25
3) min_samples_split : from 2 to 5
4) min_samples_leaf : from 1 to 5.

```
# Create dictionary for parameters
    param_dist = {'n_estimators': randint(50,500),
                  'max_depth': randint(3,25),
                 'min_samples_split': randint(2,5),
                 'min_samples_leaf': randint(1,3)}
# Create a random forest classifier
rf_tune = RandomForestClassifier()
# Use random search to find the best hyperparameters
rand_search = RandomizedSearchCV(rf, param_distributions = param_dist, n_iter=15, cv=5)
```

The best parameters for the model after doing a random search are:
n_estimators = 340, max_depth = 21, min_samples_split = 4, min_samples_leaf = 2 The confusion matrix after optimizing the model is: Though the F1 score is still the same, from the confusion matrix the number of false positives is reduced by

	Predict	
Actual	0	1
0	28186	1282
1	4122	4269
F1 Score	0.91	0.61

Table 5: Confusion Matrix

5% and number of false negatives is reduced by approximately 1%. The AOC value is increased from 0.88 to 0.89. This is not a significant improvement from the initial setup, and the reason for high false negatives is due to highly imbalanced data.

5 Discussion

Each model has its strengths and weaknesses. Logistic regression often requires additional feature engineering to improve model performance due to its sensitivity to outliers and potential redundancies introduced through multicollinearity of the features. The model does assume linearity in the data, however can be worked around using polynomial features. This however may cause the model to over fit and reduce its accuracy on a general dataset. Similarly for ANN, Neural network models have strong adaptability and learning ability with complex nonlinear data,also it can automatically extract features from raw data. However, typically require large amounts of data for training and insufficient data may lead to over-fitting. Also it sensitive to noise and outliers so that low data quality can badly influence model performance. In contrast, the Random Forest model works well for high dimensional data with good accuracy but it is susceptible to class imbalance as we see in the result and computationally expensive to do an exhaustive search for hyper tuning for larger trees. There are additional implications to consider when choosing the right model which go beyond accuracy metrics, including the business objectives, model complexity, data availability and computational cost of the model. Assessing whether the model aligns with the goals of the business and if the business can support it financially and technically is an important first consideration.

5.1 Performance

We found that all models tested outputted similar performance, but with a noticeable difference in the AUC and F1-scores. In comparing the models' performance, we specifically look at the following metrics: Confusion Matrix, F1-score, and the AUL-ROC curve. The accuracy of each model is also assessed, but is only utilised as a reference statistic, with more focus being focused on class distinction.

5.1.1 Confusion Matrix and F1-Score

Using a test set size of 37,859, the F1-scores generated for each model with confusion matrices 3, 5, and 1, are (0.9, 0.56), (0.91, 0.61) and (0.9, 0.53), for the Logitisc Regression model, Random Forest model and ANN models, in the format of (No, Yes). Here, we focus on the F1-Score for the positive class ("Yes"). These results indicate that the Random Forest model performs better than the other models, with a significant ≥ 0.5 difference. This is further supported by the accuracies of the models which are 83.68 %, 84.14 % and 85.4 % for ANN, LR, and Random Forest respectively.

5.1.2 AUC-ROC

With the AUC-ROC curve, the higher the AUC value indicates a good performance in distinguishing between binary classes. A curve that is closer to the central line, indicates that the model is no better than randomly guessing between the classes. Comparing the AUC-ROC graphs for each model (Figures 2, 3, 4), we can see that the Logistic Regression model has a much smoother and greater bend than the ANN model with a score of 0.81. However, the Random Forest model produced a much higher score of 0.89 indicating good performance in distinguishing positive and negative values.

6 Conclusion

The results of the report indicate that Random Forest is the most appropriate model for this dataset. While the performance for each model was relatively close, the Random Forest model can more reliably distinguish between positive classes, resulting in reduced False Negatives in comparison to the other models. This decision is supported through the comparison of both the AUC Score and the F1-Score. The Random Forest algorithm outperforms the other models in each of these statistics, demonstrating an accuracy of 86%. Further research could be done into the use of Neural Networks, exploring additional models in this particular niche. The flexibility of customisation and adjustment that can be made to a neural network allows for a much more in-depth exploration into better performing models. In addition, alternative models of the Random Forest model should be explored and tested, including XGBoost. In relation to Logistic Regression models, it would be beneficial to explore more sophisticated feature engineering and selection techniques to better enhance model training and accuracy.

[1]

[1] The values presented in the report might not represent exactly the values in the python notebooks.

References

[1] Australian stations measuring total monthly rainfall. http://www.bom.gov.au/climate/data/lists_by_element/alphaAUS_139.txt. Accessed: 2024-05-05.

[2] Rain in australia. https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package. Accessed: 2024-03-06.

A Pseudo-code

A.1 Logistic Regression

Step-by-step procedure for the Logistic Regression algorithm.

1. Parameter initialisation: settings up weights and bias

The weights are a 1-d vector of zeros or random small values
$\mathbf{w} \leftarrow$ initialize_weights()
This is a random scalar bias
$b \leftarrow 0$

2. Data processing: normalising data and splitting into test/train sets.

X_train, X_test, y_train, y_test $\leftarrow$ preprocess_and_split_data($\mathbf{X}, \mathbf{y}$)

3. Training: performing gradient descent to reach an optimal minima

for epoch in range(epochs):
    y_train_pred $\leftarrow$ sigmoid(X_train $\cdot$ $\mathbf{w} + b$)

    # Compute the cost function
    cost $\leftarrow -\frac{1}{m}\sum$(y_train * log(y_train_pred) + (1−y_train) * log(1−y_train_pred))

    # Compute gradients for weights and bias
    dw $\leftarrow \frac{1}{m}\sum$(X_train * (y_pred_train - y_train))
    d$b \leftarrow \frac{1}{m}\sum$(y_pred_train - y_train)

    # Update parameters using learning rate α
    $\mathbf{w} \leftarrow \mathbf{w} - \alpha *$ dw
    $b \leftarrow b - \alpha *$ db

4. Make predictions: the optimal weights and bias are now used to predict future samples. We also use the set threshold for binary classification.

y_pred_prob $\leftarrow$ sigmoid(X_test $\cdot$ $\mathbf{w} + b$)
y_pred_class $\leftarrow$ (y_pred_prob $\geq$ 0.5) ? 1 : 0

5. Perform model evaluation: here we do check performance measurements such as F1-Score, AUC-ROC, precision recall, etc.

accuracy $\leftarrow$ calculate_accuracy(y_test, y_pred_class)
print("Accuracy:", accuracy)

6. Hyper-parameter tuning: with the libraries such as sklearn, we have many parameters we can adjust to maximise the performance of our model.

A.2 Random Forest

Steps for Random Forest

```
1.  Function :  Train_Random_Forest(data, num_trees)
                Input - data:  dataset of features and target variable
                     num_trees:  Number of decision trees

2.  Initialize empty forest :  Forest = [ ]

3.  Create Decision Trees
    For i in range(num_trees)
      Take subset of data with replacement using input data
      Call function create_decision_tree(subset)
      Forest.append(Tree)

4.  Function:  create_decision_tree(data)
       If all data belong to same class:
           return leaf with class label
       If no features are left:
           return leaf node with majority class label
       Check best split:
           For each attribute in data:
               calculate Information Gain <- IG(Y|X) = H(Y) - H(Y|X)
           return node

5.  Function:  predict(Forest, new_data_point)
       Input - Forest:  trained random forest
       new_data_point:  Single data point with all features
       For each tree in Forest:
           Prediction <- Traverse tree till it reaches leaf
       return majority vote
```

A.3 Artificial Neural Network

The pseudo-code to illustrate algorithm are as follow:

Neural Network Model Generation:
1. Initialize the network structure including input, multiple hidden layers, and output layer.
2. Define the loss function and optimizer.

Training Process:
For each epoch:
a. For each batch in the training dataset:
i. Forward Propagation: Feed data through the network and compute the output.
ii. Compute Loss: Calculate the loss using the network's output and the actual labels.
iii. Backpropagation: Compute gradients of the loss and update the network weights using the optimizer.

B Dataset Dictionary

Name	Description	Datatype
Date	The date of observation.	datetime
Location	The common name of the location of the weather station.	string
MinTemp	The minimum temperature in degrees celsius.	float
MaxTemp	The maximum temperature in degrees celsius.	float
Rainfall	The amount of rainfall recorded for the day in mm.	float
Evaporation	The so-called Class A pan evaporation (mm) in the 24 hours to 9am.	float
Sunshine	The number of hours of bright sunshine in the day.	integer
WindGustDir	The direction of the strongest wind gust in the 24 hours to midnight.	string
WindGustSpeed	The speed (km/h) of the strongest wind gust in the 24 hours to midnight.	integer
WindDir9am	Direction of the wind at 9am.	string
WindDir3pm	Direction of the wind at 3pm.	string
WindSpeed9am	Wind speed (km/hr) averaged over 10 minutes prior to 9am.	integer
WindSpeed3pm	Wind speed (km/hr) averaged over 10 minutes prior to 3pm.	integer
Humidity9am	Humidity (percent) at 9am.	integer
Humidity3pm	Humidity (percent) at 3pm.	integer
Pressure9am	Atmospheric pressure (hpa) reduced to mean sea level at 9am.	float
Pressure3pm	Atmospheric pressure (hpa) reduced to mean sea level at 3pm.	float
Cloud9am	Fraction of sky obscured by clouds at 9am. This is measured in "oktas", which are a unit of eighths. It records how many eighths of the sky are obscured by clouds. A 0 measure indicates completely clear sky whilst an 8 indicates that it is completely overcast.	integer
Cloud3pm	Fraction of sky obscured by clouds (in "oktas": eighths) at 3pm. See Cload9am for a description of the values.	integer
Temp9am	Temperature (degrees C) at 9am.	float
Temp3pm	Temperature (degrees C) at 3pm.	float
RainToday	Boolean: 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0.	bool
RainTomorrow	The amount of next day rain in mm. Used to create the response variable RainTomorrow. A kind of measure of the "risk".	bool

Table 6: Dataset overview