Maltepe University
Computer Engineering Department

Data Mining Project

Advisor:
Assist. Prof. Dr. Volkan Tunali

Students:
180706001 - Sabahattin Emirhan Sönmez
200704701 - Paula-Diana Băcîrcea

İstanbul, 2021

Maltepe University
Computer Engineering Department

Rain Prediction in Australia

Advisor:
Assist. Prof. Dr. Volkan Tunali

Students:
180706001 - Sabahattin Emirhan Sönmez
200704701 - Paula-Diana Băcîrcea

İstanbul, 2021

Table of Contents

Introduction

The purpose of this project is to design a predictive model capable of predicting whether or not it's going to rain in a certain location in Australia based on the observations of the current weather.

The Problem

Taking into consideration the wildfires that had taken place all across Australia last year, it is important to predict whether or not it's going to rain in order to start taking action as early as possible in case of a potential catastrophe.

Dataset

Data Analysis

The dataset we picked for this project is called **Rain in Australia** and it was provided by the Kaggle user **Joe Young**. It contains information of about 10 years of daily weather observations from 49 different locations across Australia. The dataset consists of 145460 observations and no duplicates. Also, there are 23 variables involved in this dataset.

Numerical variables

- **Interval variables**
  - **Date** - the date of observation
  - **MinTemp** - the minimum temperature (measured in °C)
  - **MaxTemp** - the maximum temperature (measured in °C)
  - **Humidity9am** - the humidity at 9am (percentage)
  - **Humidity3pm** - the humidity at 3pm (percentage)
  - **Temp9am** - the temperature at 9am (measured in °C)
  - **Temp3pm** - the temperature at 3pm (measured in °C)
  - **Cloud9am** - a scale from 0 to 8, 0 - completely clear sky, 8 - completely overcast
  - **Cloud3pm** - a scale from 0 to 8, 0 - completely clear sky, 8 - completely overcast
- **Ratio variables**
  - **Rainfall** - the amount of rainfall (measured in mm)
  - **Evaporation** - the evaporation (measured in mm)
  - **Sunshine** - the number of hours of bright sunshine in the day
  - **WindGustSpeed** - the speed of the strongest wind
  - **WindSpeed9am** - the average speed of the wind measured at 9am (measured in km/hr)

- **WindSpeed3pm** - the average speed of the wind measured at 3pm (measured in km/hr)
- **Pressure9am** - atmospheric pressure recorded at 9am (measured in hpa)
- **Pressure3pm** - atmospheric pressure recorded at 3pm (measured in hpa)

Categorical variables

- **Nominal variables**
    - **Location** - the location of the weather station
    - **WindGustDir** - the direction of the strongest wind
    - **WindDir9am** - the direction of the wind at 9am
    - **WindDir3pm** - the direction of the wind at 3pm

There are also two **boolean variables**:

RainToday - it's 1 if the Rainfall value exceeds 1mm, and 0 otherwise
RainTomorrow - the target variable to be predicted

Missing Values

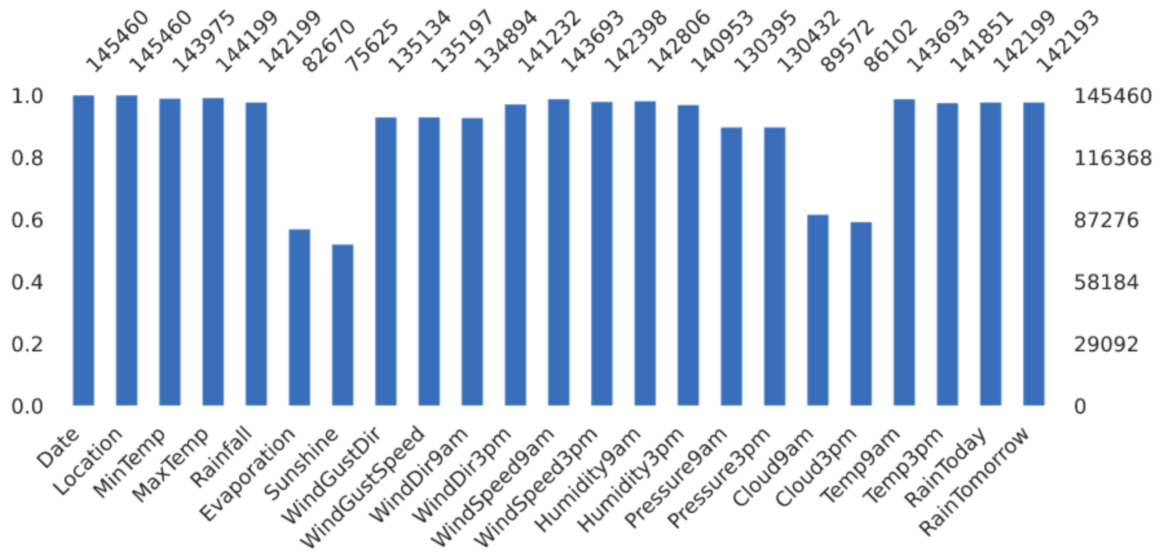

Figure 1. A simple visualization of nullity by column

During the analysis, we've discovered the following:

- MinTemp has 1485 (1.0%) missing values
- Rainfall has 3261 (2.2%) missing values
- Evaporation has 62790 (43.2%) missing values
- Sunshine has 69835 (48.0%) missing values
- WindGustDir has 10326 (7.1%) missing values
- WindGustSpeed has 10263 (7.1%) missing values

- WindDir9am has 10566 (7.3%) missing values
- WindDir3pm has 4228 (2.9%) missing values
- WindSpeed9am has 1767 (1.2%) missing values
- WindSpeed3pm has 3062 (2.1%) missing values
- Humidity9am has 2654 (1.8%) missing values
- Humidity3pm has 4507 (3.1%) missing values
- Pressure9am has 15065 (10.4%) missing values
- Pressure3pm has 15028 (10.3%) missing values
- Cloud9am has 55888 (38.4%) missing values
- Cloud3pm has 59358 (40.8%) missing values
- Temp9am has 1767 (1.2%) missing values
- Temp3pm has 3609 (2.5%) missing values
- RainToday has 3261 (2.2%) missing values
- RainTomorrow has 3267 (2.2%) missing values

Among the columns with missing values, Evaporation, Sunshine, Cloud9am, and Cloud3pm were the columns with the most missing values (>35%).

Correlations

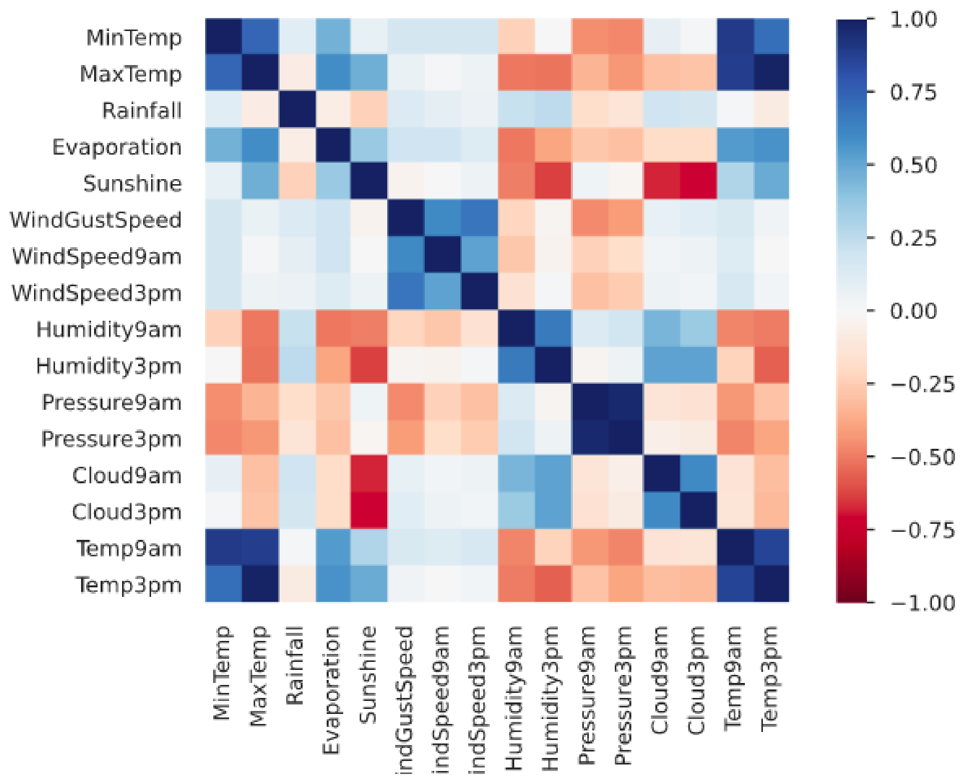

Figure 2. Correlation matrix using Pearson’s r coefficient

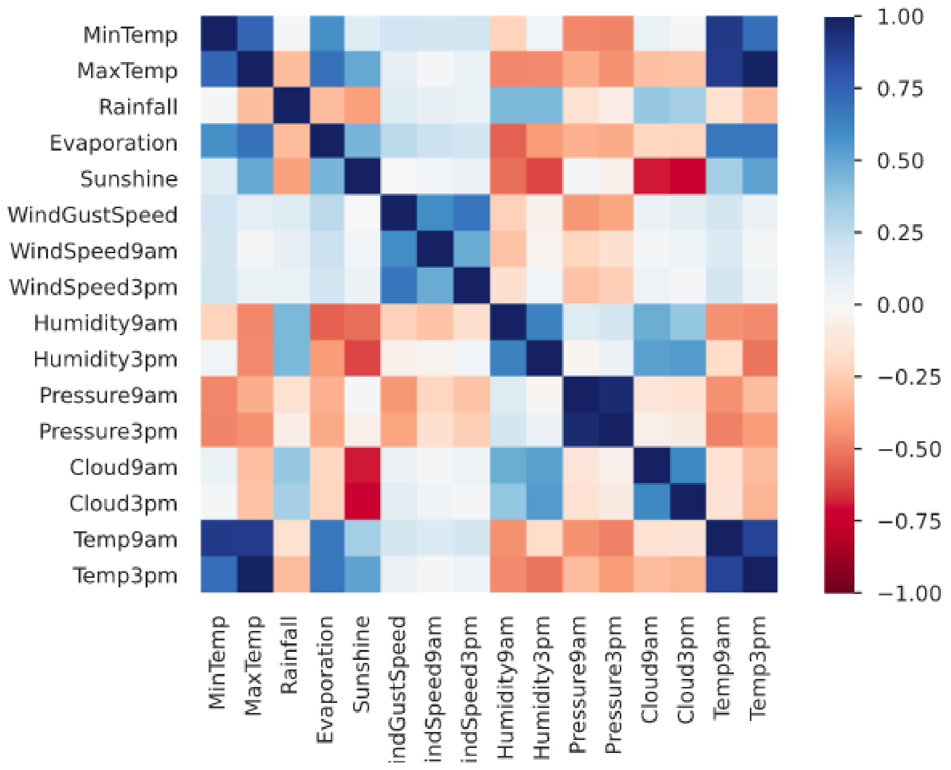

Figure 3. Correlation matrix using Spearman's ρ coefficient

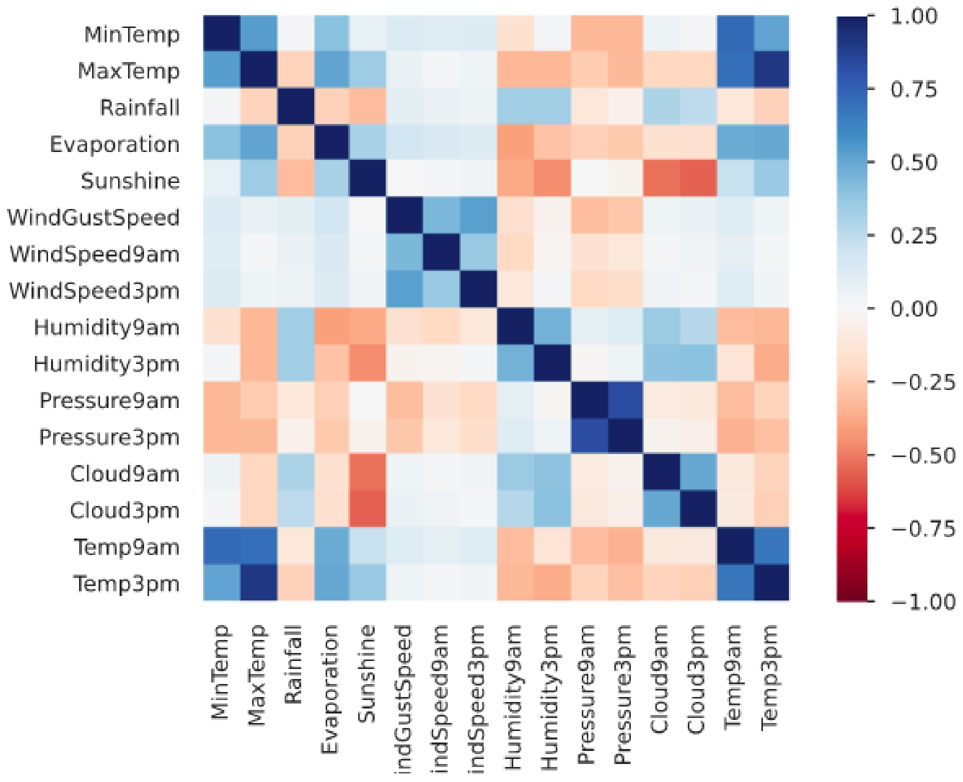

Figure 4. Correlation matrix using Kendall's τ coefficient

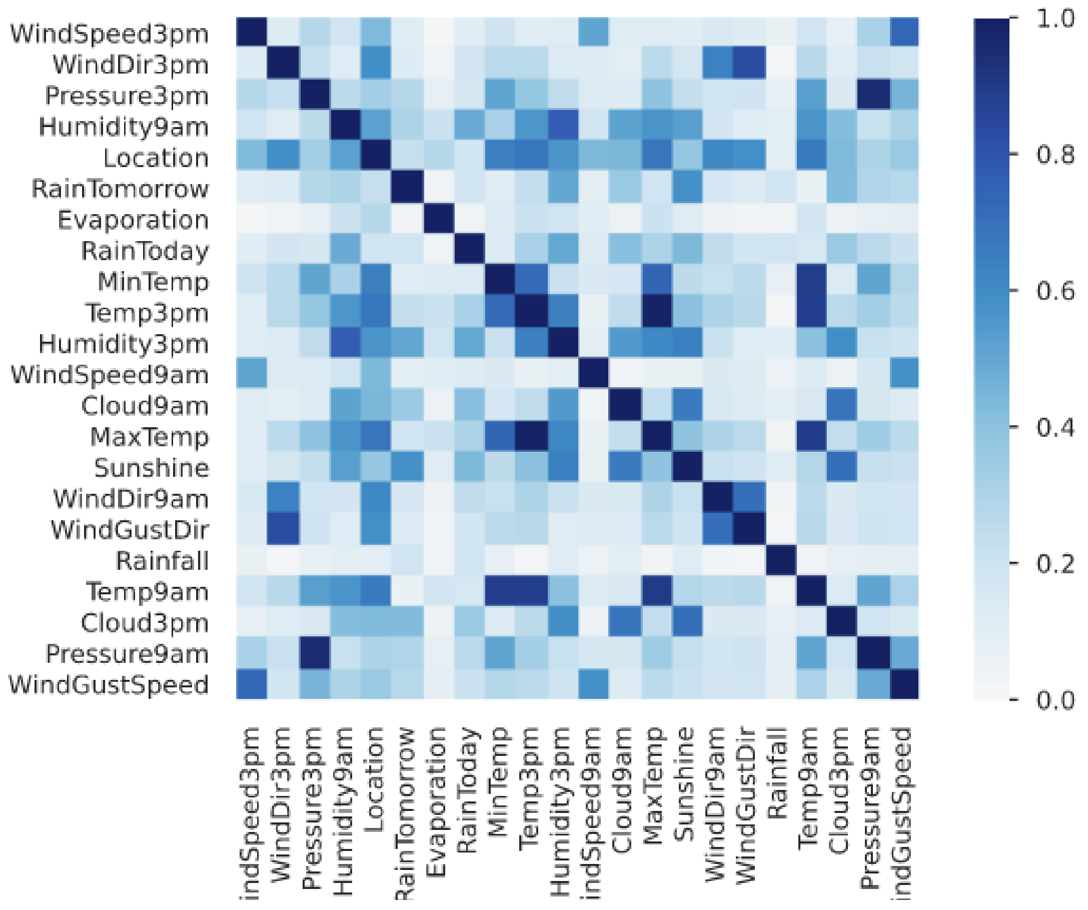

Figure 5. Correlation matrix using Phik (φk) coefficient

To analyse the data, we used four different correlation matrices and in each case we reached the same results:

- MinTemp is highly correlated with Temp9am
- MaxTemp is highly correlated with Temp3pm
- Pressure9am is highly correlated with Pressure3pm
- Pressure3pm is highly correlated with Pressure9am

From these correlations, we've concluded that the minimum temperature usually occurs at 9am, and the maximum temperature usually occurs at 3pm. We've also concluded that the pressure doesn't really change throughout the day.

Zeros

During the analysis, we've discovered the following:

- Rainfall has 91080 (62.6%) zeros
- Sunshine has 2359 (1.6%) zeros
- WindSpeed9am has 8745 (6.0%) zeros
- Cloud9am has 8642 (5.9%) zeros
- Cloud3pm has 4974 (3.4%) zeros

Data Preprocessing

In order to preprocess the data, we've followed the following steps:

1. We dropped all the rows that had a missing value for the RainTomorrow attribute because it wouldn't have been possible to use those rows for training the models.
2. We updated the missing values of the RainToday attribute based on the value of the Rainfall attribute, so if the Rainfall is greater than 1.0 mm, RainToday is equal with Yes and if the Rainfall is less than or equal to 1.0 mm, RainToday is equal with No.
3. If there were some rows where both the Rainfall and the RainToday were missing, we decided to drop those rows as well.
4. We mapped the 'No' and 'Yes' values of the binary columns to 0 and 1 values , since it is easier for the model to process them.
5. We dropped the columns that were highly correlated.
6. We encoded the categorical columns using the `sklearn.preprocessing.LabelEncoder()` class
7. To fill the missing values for the rest of the columns, we did the following:
    a. for the categorical attributes, we filled the missing values with the most frequent value for that particular column.
    b. for the numerical attributes, we filled the missing values with the mean value for that particular column.
8. Separate the data into train and test data.

Data Mining Algorithms Used

Naive Bayes

Naive Bayes algorithm can be defined as a supervised classification algorithm which is based on Bayes theorem with an assumption of independence among features.

We have decided to use this algorithm because:

- It is easy to build and particularly useful for very large data sets.
- It is known for its simplicity, being capable to outperform even highly sophisticated classification methods

For our model, we have set the following hyperparameter:

- **`var_smoothing`**, which is the portion of the largest variance of all features that is added to variances for calculation stability.

Logistic Regression

It's a classification algorithm that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.

We have decided to use this algorithm because:

- It is easier to implement, interpret, and very efficient to train.
- It makes no assumptions about distributions of classes in feature space.
- It is very fast at classifying unknown records.

For our model, we have set the following hyperparameters:

- **`solver`**, which is the algorithm to use in the optimization problem.
- **`max_iter`**, which is the maximum number of iterations taken for the solvers to converge.

K-Nearest Neighbours

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data.

We have decided to use this algorithm because:

- It has a quick calculation time
- It is a simple algorithm – to interpret

- It is versatile – useful for regression and classification
- It has a high accuracy – you do not need to compare with better-supervised learning models
- It makes no assumptions about the data – no need to make additional assumptions, tune several parameters, or build a model. This makes it crucial in nonlinear data cases.

For our model, we have set the following hyperparameters:

- **`n_neighbors`**, which is the number of neighbors required for each sample. The default is the value passed to the constructor.
- **`p`**, which is the power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

Linear Support Vector

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

SVMs are one of the most robust prediction methods, being based on statistical learning frameworks or VC theory proposed by Vapnik (1982, 1995) and Chervonenkis (1974).

SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

The Linear Support Vector is similar to SVC with parameter `kernel='linear'`, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

We have decided to use this algorithm because:

- It works relatively well when there is a clear margin of separation between classes.
- It is more effective in high dimensional spaces.
- It is effective in cases where the number of dimensions is greater than the number of samples.
- It is relatively memory efficient.

For our model, we have set the following hyperparameters:

- **`loss`**, which specifies the loss function. ‘hinge’ is the standard SVM loss (used e.g. by the SVC class) while ‘squared_hinge’ is the square of the hinge loss. The combination of penalty='l1' and loss='hinge' is not supported.
- **`max_iter`,** which is the maximum number of iterations to be run.

Decision Tree

The Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can be used to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record’s attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

We have decided to use this algorithm because:

- Decision trees often mimic human level thinking so it's so simple to understand the data and make some good interpretations.
- Decision trees actually make you see the logic for the data to interpret(not like black box algorithms like SVM,NN,etc..)

For our model, we have set the following hyperparameter:

- **`criterion`,** which is the function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

Bagging Decision Tree

Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here the idea is to create several subsets of data from training samples chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.

We have decided to use this algorithm because:

- It allows many weak learners to combine efforts to outdo a single strong learner.
- It helps in the reduction of variance, hence eliminating the overfitting of models in the procedure.

For our model, we have set the following hyperparameter:

- **`max_features`**, which is the number of features to draw from X to train each base estimator (without replacement by default, see bootstrap_features for more details).

Random Forest

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is: A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions.

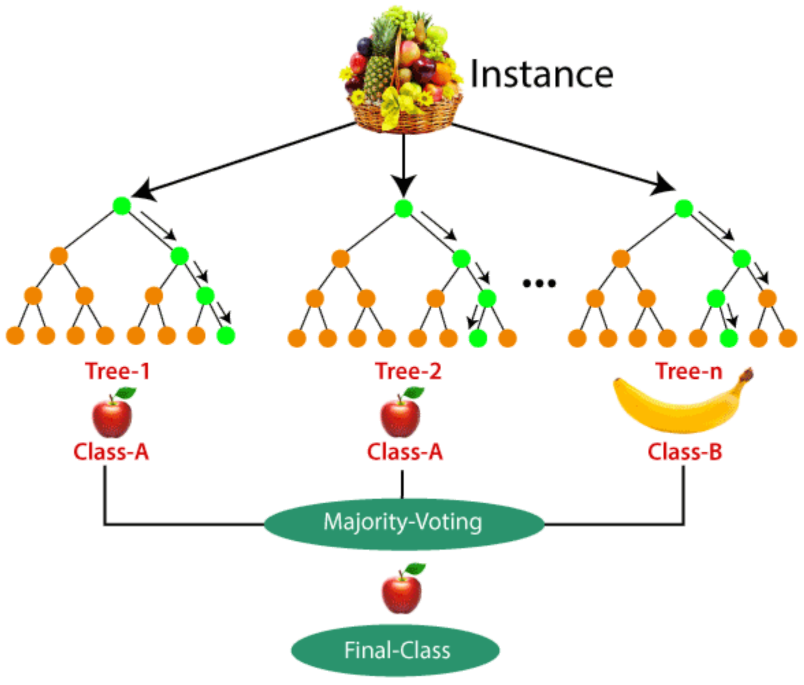

Figure 6. Example of how a Random Forest classifier works

We have decided to use this algorithm because:

- It reduces overfitting in decision trees and helps to improve the accuracy
- It is flexible to both classification and regression problems
- It works well with both categorical and continuous values
- It automates missing values present in the data
- It removes the need of normalising the data because it uses a rule-based approach.

For our model, we have set the following hyperparameters:

- **`max_depth`,** which is the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **`max_features`,** which is the number of features to consider when looking for the best split

Experiments and Results

In order to determine the best hyperparameters for each of the models we used the `sklearn.model_selection.GridSearchCV` function which takes as input a space of parameters and determines the ones that achieve the highest score.

Best hyperparameters for each model

Naive Bayes

The function determined that the best value for the `var_smoothing` hyperparameter is `1e-5`, which results in a score of `0.742` on the training set.

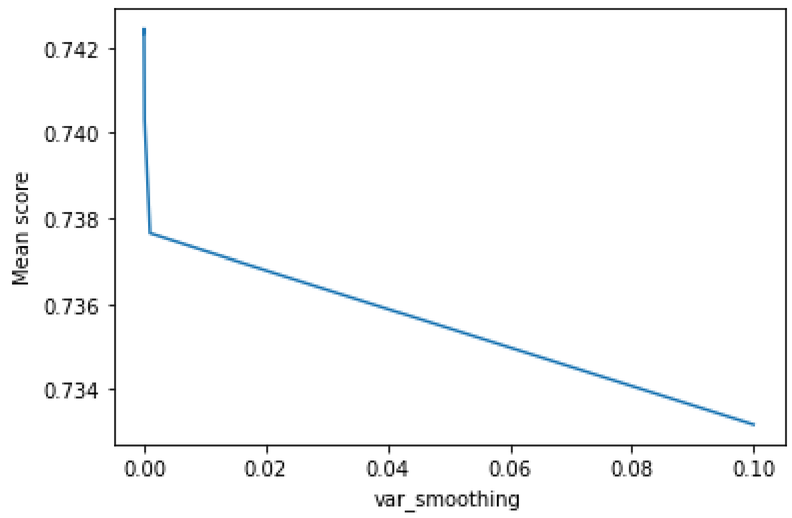

Figure 7. The Graph of the relation between var_smoothing and the mean score

This model predicted the labels of the test set with an accuracy of 0.747.

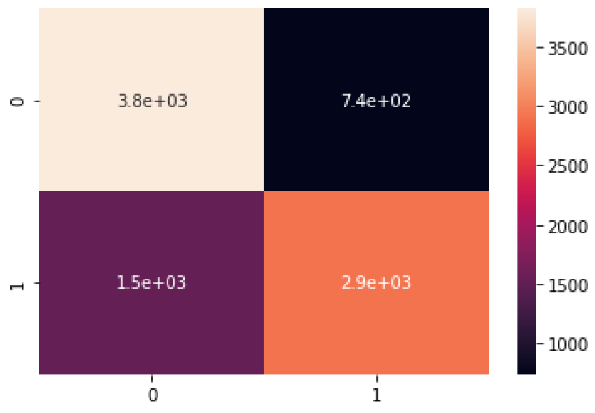

Figure 8. The Confusion Matrix for the Naive Bayes classifier

Logistic regression

The function determined that the best values for the `solver` and `max_iter` hyperparameters are `sag`, respectively `2000`, which results in a score of `0.775` on the training set.

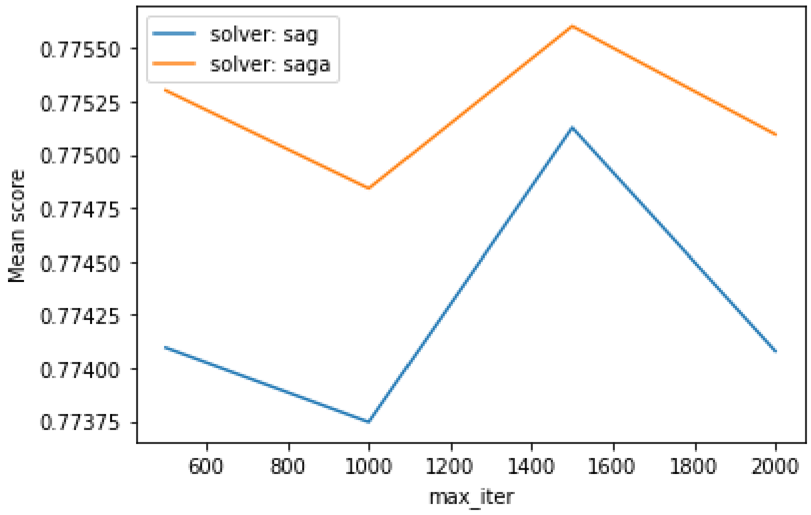

Figure 9. The Graph of the relation between max_iter, solver, and the mean score

This model predicted the labels of the test set with an accuracy of 0.778.

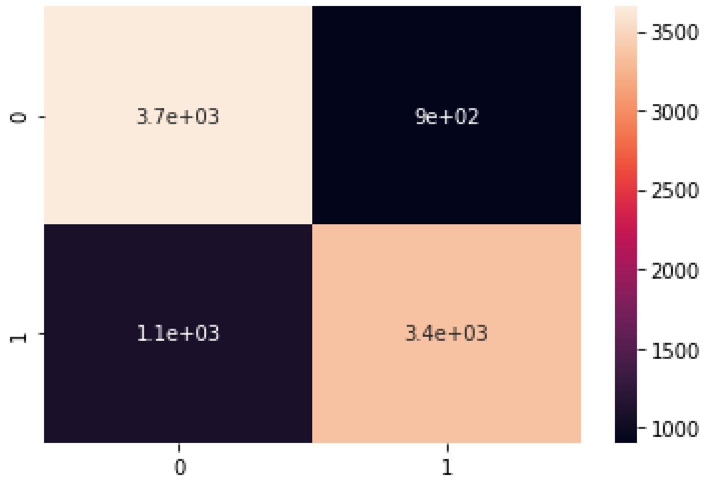

Figure 10. The Confusion Matrix for the Logistic Regression classifier

K-Nearest Neighbours

The function determined that the best values for the `n_neighbors` and `p` hyperparameters are `15`, respectively `1`, which results in a score of `0.788` on the training set.

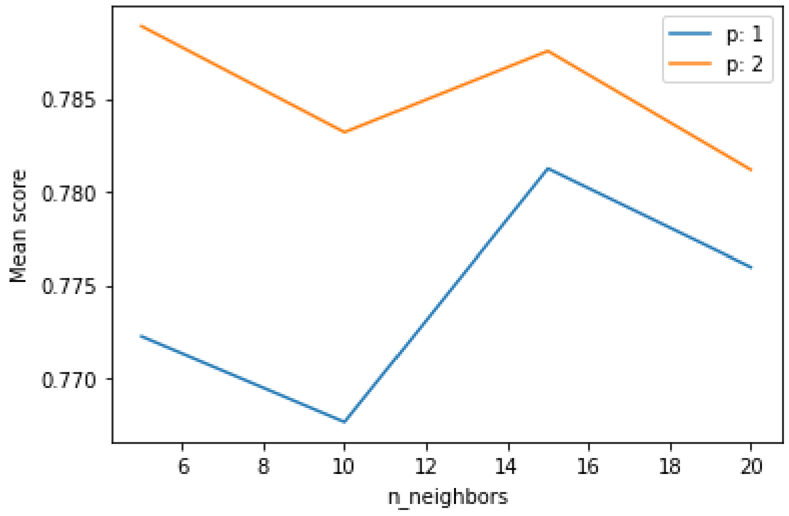

Figure 11. The Graph of the relation between n_neighbors, p, and the mean score

This model predicted the labels of the test set with an accuracy of 0.789.

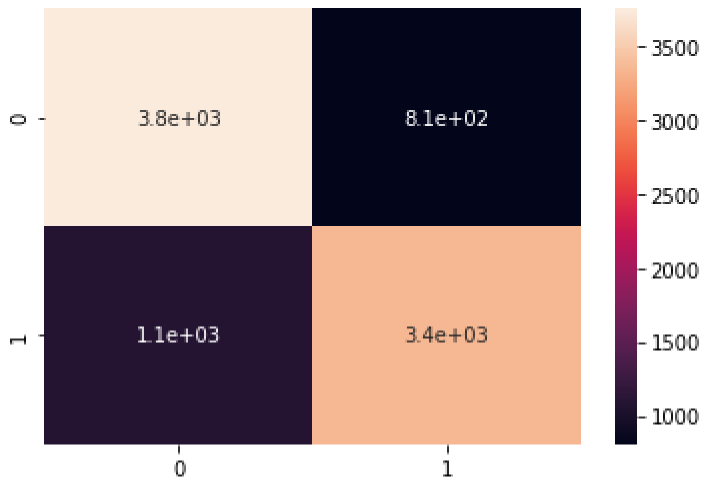

Figure 12. The Confusion Matrix for the K-Nearest Neighbours classifier

Linear Support Vector

The function determined that the best values for the `loss` and `max_iter` hyperparameters are `hinge`, respectively `2000`, which results in a score of `0.678` on the training set.

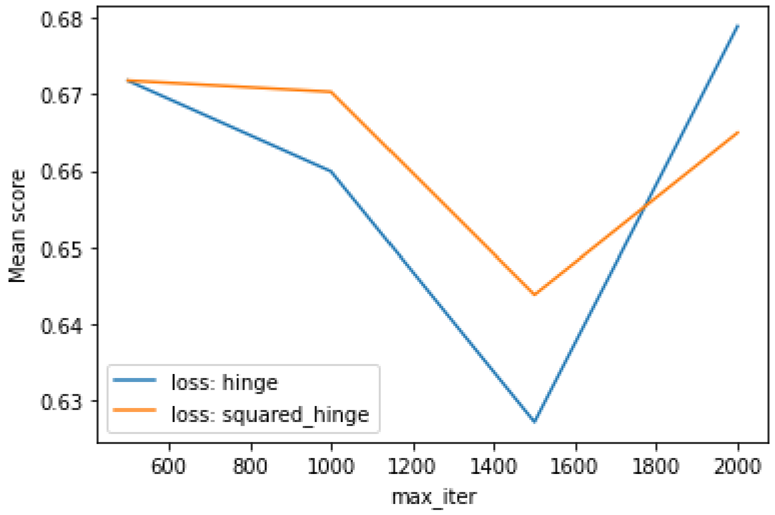

Figure 13. The Graph of the relation between max_iter, loss, and the mean score

This model predicted the labels of the test set with an accuracy of 0.722.

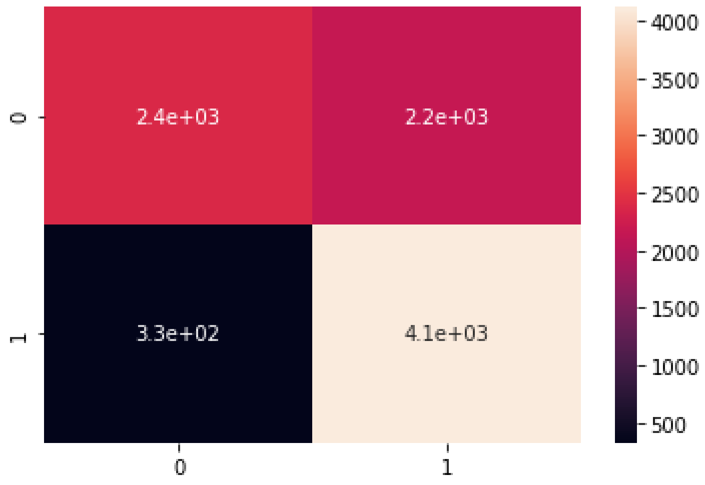

Figure 14. The Confusion Matrix for the Linear Support Vector classifier

Decision Tree

The function determined that the best value for the `criterion` hyperparameter is `entropy`, which results in a score of `0.715` on the training set.

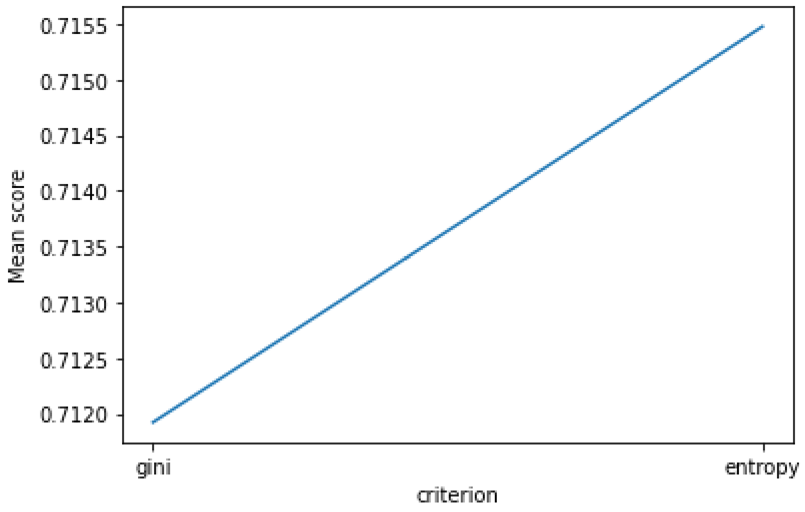

Figure 15. The Graph of the relation between criterion and the mean score

This model predicted the labels of the test set with an accuracy of 0.709.

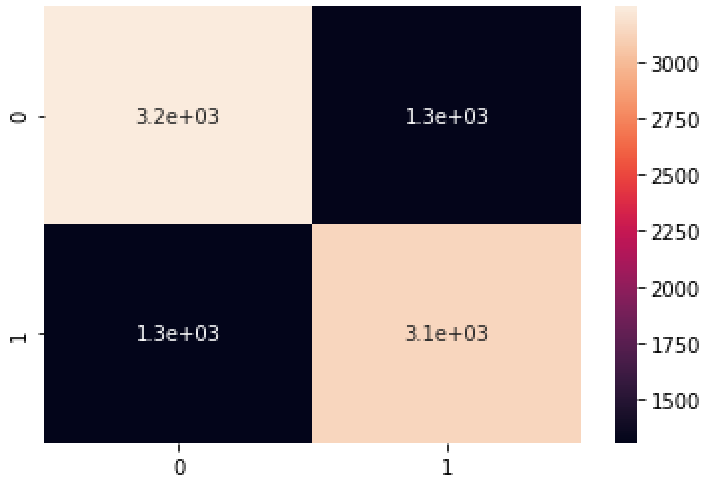

Figure 16. The Confusion Matrix for the Decision Tree classifier

Bagging Decision Tree

The function determined that the best value for the `max_features` hyperparameter is `9`, which results in a score of `0.748` on the training set.

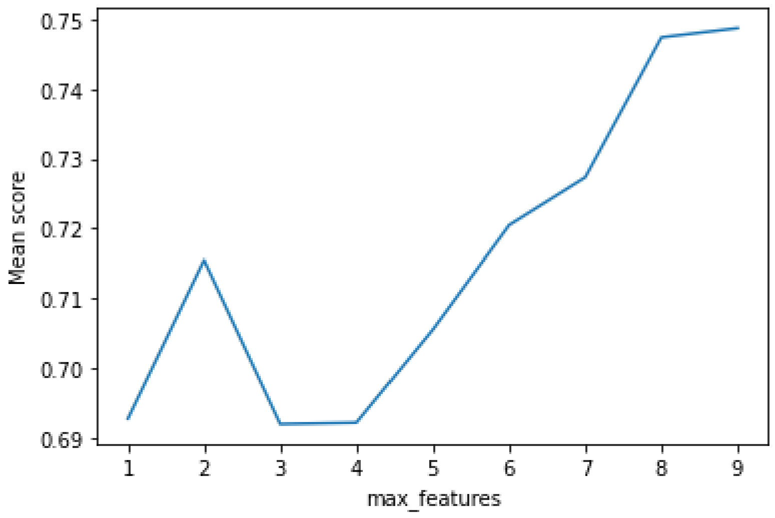

Figure 17. The Graph of the relation between max_features and the mean score

This model predicted the labels of the test set with an accuracy of 0.754.

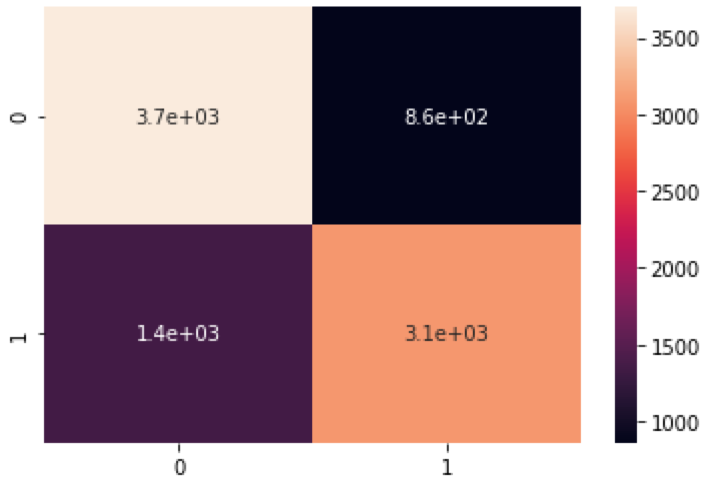

Figure 18. The Confusion Matrix for the Bagging Tree classifier

Random Forest

The function determined that the best values for the max_depth and max_features hyperparameters are `10`, respectively `auto`, which results in a score of `0.747` on the training set.

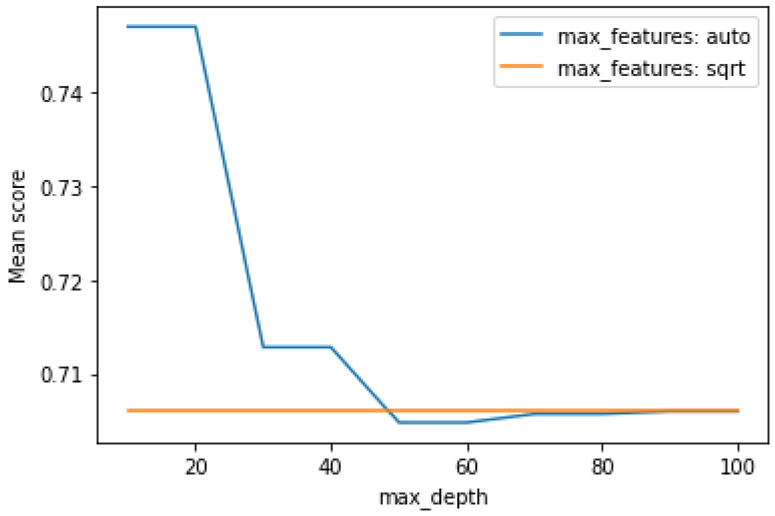

Figure 19. The Graph of the relation between max_depth, max_features, and the mean score

This model predicted the labels of the test set with an accuracy of 0.744.

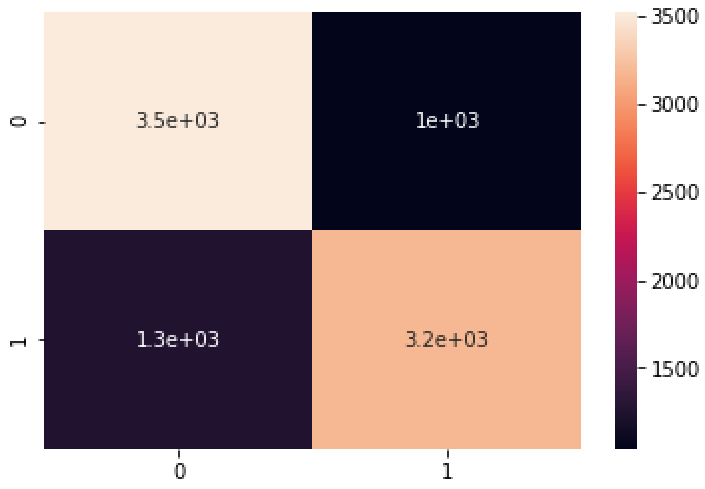

Figure 20. The Confusion Matrix for the Random Forest classifier

The Results

Model	**Train Score**	**Test Score**
Naive Bayes	0.742	0.747
Logistic Regression	0.775	0.778
K-Nearest Neighbors	0.788	0.789
Linear Support Vector Classification	0.678	0.722
Decision Tree	0.715	0.709
Bagging Decision Tree	0.748	0.754
Random Forest	0.747	0.744

Conclusion

The model which performed the best was the **K-Nearest Neighbours Classifier** with a score of `0.789`, while the model which performed the worst was the **Decision Tree Classifier** with a score of `0.709`. It is also worth mentioning that almost all the models performed a little better on the test data than on the train data, meaning the models don't have the tendency of overfitting.

Appendix

The entire code can be visualized **here**.

```
# -*- coding: utf-8 -*-
"""CEN 416 Term Project Notebook - Sabahattin Emirhan Sönmez and
Paula-Diana Băcîrcea.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1uE62bAulBYjd8ZKdIebm3bHh7
uQLyQWi
"""
! pip install
https://github.com/pandas-profiling/pandas-profiling/archive/maste
r.zip

"""# Importing the data

"""

import pandas as pd
url =
'https://raw.githubusercontent.com/dianapaula19/rain-prediction-au
stralia/master/data/weatherAUS.csv'
data = pd.read_csv(url)

"""# Analysing the data"""
```

```
data.info()

import pandas_profiling
data.profile_report(title="Rain in Australia Data Analysis",
explorative=True)

"""# Preprocessing the Data"""

columns = {i: col for i, col in enumerate(data.columns)}
num = [2, 3, 4, 5, 6, 8, 11, 12, 13, 14, 15, 16, 19, 20]
cat = [1, 7, 9, 10, 17, 18]
print(columns)

"""## Remove all the rows where the RainTomorrow value is missing
"""

for col in [21, 22]:
  data = data[data[columns[col]].notna()]

"""## Remove the duplicates"""

data.drop_duplicates(inplace=True)

"""## Map the 'Yes' and 'No' values in the RainToday and
RainTomorrow columns"""

for c in [21, 22]:
  data[columns[c]].replace({'Yes': 1, 'No': 0, 'YesS': 1},
inplace=True)

"""## Fill the missing values in the RainToday column based on the
Rainfall columns"""

new_rain_col = []
for x, y in zip(data[columns[4]], data[columns[21]]):
  if x != 'nan' and y == 'nan':
    if x > 1.0:
      new_rain_col.append(0)
    else:
        new_rain_col.append(1)
  else:
    new_rain_col.append(y)
```

```
data[columns[21]] = new_rain_col

"""## Replace the missing values"""

for col in num: # numerical attributes
  new_val = round(data[columns[col]].mean(), 1)
  data[columns[col]].fillna(new_val, inplace=True)

for col in cat: # categorical attributes
  new_val = data[columns[col]].value_counts().argmax()
  data[columns[col]].fillna(new_val, inplace=True)

"""## Encoding the categorical columns"""

from sklearn import preprocessing as pre
le = pre.LabelEncoder()
for col in [columns[c] for c in cat]:
  data[col] = le.fit_transform(data[col].astype(str))

"""## Drop the highly correlated columns"""

data.drop([columns[c] for c in [0, 13, 19, 20]], axis=1,
inplace=True)

"""## Check for any missing values"""

print(data.isna().sum())
print("Number of columns: " + str(len(data.columns)))

"""## Separate the data into train and test data
21000 rows - train

9000 rows - test
"""

from sklearn.model_selection import train_test_split
df = data.groupby('RainTomorrow').sample(n=15000, replace=False)
print(df.shape[0])
print(df['RainTomorrow'].unique())
x = df[df.columns[0:18]]
y = df[df.columns[18]]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=42)

"""# Models"""

from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
random_state=42)

"""## Naive Bayes"""

from sklearn.naive_bayes import GaussianNB
gnb_params = {
    'var_smoothing': [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7,
1e-8, 1e-9]
}
gnb = GaussianNB()
search = GridSearchCV(gnb, gnb_params, scoring='accuracy',
n_jobs=-1, cv=cv)
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores = np.array(result.cv_results_['mean_test_score'])

plt.plot(gnb_params['var_smoothing'], scores)
plt.xlabel('var_smoothing')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))

"""## Logistic Regression"""
```

```
from sklearn.linear_model import LogisticRegression
lr_params = {
    'solver': ['sag', 'saga'],
    'max_iter': [500, 1000, 1500, 2000]
}
lr = LogisticRegression(random_state=42)
search = GridSearchCV(lr, lr_params, scoring='accuracy',
n_jobs=-1, cv=cv)
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores =
np.array(result.cv_results_['mean_test_score']).reshape(len(lr_par
ams['solver']), len(lr_params['max_iter']))

for ind, i in enumerate(lr_params['solver']):
    plt.plot(lr_params['max_iter'], scores[ind], label='solver: '
+ str(i))

plt.legend()
plt.xlabel('max_iter')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))

"""## K-Nearest Neighbors"""

from sklearn.neighbors import KNeighborsClassifier
knn_params = {
    'n_neighbors': [5, 10, 15, 20],
    'p': [1, 2]
}
knn = KNeighborsClassifier()
search = GridSearchCV(knn, knn_params, scoring='accuracy',
n_jobs=-1, cv=cv)
```

```
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores =
np.array(result.cv_results_['mean_test_score']).reshape(len(knn_pa
rams['p']), len(knn_params['n_neighbors']))

for ind, i in enumerate(knn_params['p']):
    plt.plot(knn_params['n_neighbors'], scores[ind], label='p: ' +
str(i))

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))

"""## Linear Support Vector Classification"""

from sklearn.svm import LinearSVC
lsvc_params = {
    'loss': ['hinge', 'squared_hinge'],
    'max_iter': [500, 1000, 1500, 2000]
}
lsvc = LinearSVC(random_state=42)
search = GridSearchCV(lsvc, lsvc_params, scoring='accuracy',
n_jobs=-1, cv=cv)
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores =
np.array(result.cv_results_['mean_test_score']).reshape(len(lsvc_p
arams['loss']), len(lsvc_params['max_iter']))

for ind, i in enumerate(lsvc_params['loss']):
    plt.plot(lsvc_params['max_iter'], scores[ind], label='loss: '
+ str(i))

plt.legend()
plt.xlabel('max_iter')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))

"""## Decision Tree"""

from sklearn.tree import DecisionTreeClassifier
dt_params = {
    'criterion': ['gini', 'entropy'],
}
dt = DecisionTreeClassifier(random_state=42)
search = GridSearchCV(dt, dt_params, scoring='accuracy',
n_jobs=-1, cv=cv)
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores = np.array(result.cv_results_['mean_test_score'])
plt.plot(dt_params['criterion'], scores)
plt.xlabel('criterion')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))

"""## Bagging Decision Tree"""

from sklearn.ensemble import BaggingClassifier
```

```
bg_params = {
  'max_features': [i for i in range(1, 10)]
}
bg = BaggingClassifier(random_state=42)
search = GridSearchCV(bg, bg_params, scoring='accuracy',
n_jobs=-1, cv=cv)
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores = np.array(result.cv_results_['mean_test_score'])
plt.plot(bg_params['max_features'], scores)
plt.xlabel('max_features')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))

"""## Random Forest"""

from sklearn.ensemble import RandomForestClassifier
rf_params = {
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    'max_features': ['auto', 'sqrt']
}
rf = DecisionTreeClassifier(random_state=42)
search = GridSearchCV(rf, rf_params, scoring='accuracy',
n_jobs=-1, cv=cv)
result = search.fit(x_train, y_train)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)

scores =
np.array(result.cv_results_['mean_test_score']).reshape(len(rf_par
ams['max_features']), len(rf_params['max_depth']))

for ind, i in enumerate(rf_params['max_features']):
    plt.plot(rf_params['max_depth'], scores[ind],
label='max_features: ' + str(i))

plt.legend()
plt.xlabel('max_depth')
plt.ylabel('Mean score')
plt.show()

y_pred = result.predict(x_test)
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True)
print("The accuracy score is: " + str(accuracy_score(y_test,
y_pred)))
```

List of Figures