Bushfire Hazard Reduction Feasibility Prediction

Randompdf

Exported and Printed

February 6, 2025

0.1 Introduction

This document provides an **overview of the Bushfire Hazard Reduction Feasibility Prediction project**, which applies **machine learning techniques** to predict whether a controlled burn can be conducted safely based on historical weather data. The objective is to support decision-making in mitigating bushfire risks in Australia.

The study is based on a dataset containing **over 134,000 records with 24 variables**, covering **weather conditions, location, and seasonal factors**. A structured approach was followed, including **data preprocessing, feature engineering, class imbalance handling, and model evaluation**.

0.1.1 Exploratory Data Analysis and Clustering

As part of the initial analysis, **k-Means Clustering** was applied to identify potential patterns in the data. However, the **Silhouette Score of 0.213** indicated that clustering did not provide strong separation for classification purposes.

0.1.2 Classification Algorithms Used

Several machine learning models were employed for classification, including:
- **Decision Tree**
- **Artificial Neural Network (ANN)**
- **Gaussian Naïve Bayes (NBG)**
- **Support Vector Machine (SVM)**
- **Random Forest**

Each model was evaluated using **accuracy, precision, recall, F1-score, and confusion matrices**, with a special focus on **handling class imbalance** and optimizing performance. Among the models, **Random Forest demonstrated the highest accuracy and balanced precision-recall, making it a strong candidate for practical deployment**.

0.1.3 Access to Full Project

For those interested in further exploration, the **full code, environment setup, and dataset** are available **upon request**. If you would like access to these materials, please reach out.

This document will also be uploaded to a **public GitHub repository**, ensuring transparency and accessibility for the research and machine learning communities.

[1]:
```python
# The order of importing libraries are according to the sequence of steps taken␣
↪in our work.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import openpyxl

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split


from sklearn.neural_network import MLPClassifier

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import classification_report,␣
↪confusion_matrix,accuracy_score
from sklearn.model_selection import cross_val_score
```

[2]:
```python
# Load dataset
df = pd.read_excel("A1_Data.xlsx")
```

[3]:
```python
df.shape
```

[3]: (134692, 24)

[4]:
```python
# Get the data types of each column
column_types = df.dtypes

# Print the data types
print(column_types)
```

```
City_Name                      object
City_State                     object
Date                           object
Season                         object
Climate                        object
Minimum_Temperature           float64
Maximum_Temperature           float64
Temperature_AM                float64
Temperature_PM                float64
Wind_Direction_AM              object
Wind_Direction_PM              object
Wind_Speed_AM                 float64
Wind_Speed_PM                 float64
```

```
Max_WindGust_Direction      object
Max_WindGust_Speed         float64
Significant_Rainfall        object
Rainfall_Amount            float64
Humidity_AM                float64
Humidity_PM                float64
Atmospheric_Pressure_AM    float64
Atmospheric_Pressure_PM    float64
CloudCover_Oktas_AM          int64
CloudCover_Oktas_PM          int64
Burn_Tomorrow                int64
dtype: object
```

[5]:

```
def identify_data(df):
    # Check for missing values
    print('Missing values in the DataFrame:\n', df.isnull().sum(),'\n')
    # Check for duplicates
    print('Number of duplicated rows in the DataFrame:\n', df.duplicated().
 ↪sum(),'\n')
    # Check for unique values
    print('Number of unique values in each column of the DataFrame:\n', df.
 ↪nunique(),'\n')
    # Check for out of bound value
    stats=df.describe()
    print(stats)


    # Set the figure size to make the plot wider
    plt.figure(figsize=(16, 6))
    #Create a boxplot for each numeric column
    ax = df.boxplot()

    # Set the rotation angle for x-axis labels
    plt.xticks(rotation=45)  # You can adjust the rotation angle as needed

   # Show the plot
    plt.show()
identify_data(df)
```

```
Missing values in the DataFrame:
 City_Name                    0
City_State                   0
Date                         0
Season                       0
Climate                      0
Minimum_Temperature        582
Maximum_Temperature        305
Temperature_AM               0
```

```
Temperature_PM                0
Wind_Direction_AM          9531
Wind_Direction_PM          3558
Wind_Speed_AM              1323
Wind_Speed_PM              2452
Max_WindGust_Direction     9027
Max_WindGust_Speed         8968
Significant_Rainfall          0
Rainfall_Amount            1350
Humidity_AM                   0
Humidity_PM                   0
Atmospheric_Pressure_AM       0
Atmospheric_Pressure_PM       0
CloudCover_Oktas_AM           0
CloudCover_Oktas_PM           0
Burn_Tomorrow                 0
dtype: int64

Number of duplicated rows in the DataFrame:
 0

Number of unique values in each column of the DataFrame:
 City_Name                   48
City_State                    8
Date                       3337
Season                        4
Climate                       5
Minimum_Temperature         389
Maximum_Temperature         505
Temperature_AM              440
Temperature_PM              500
Wind_Direction_AM            16
Wind_Direction_PM            16
Wind_Speed_AM                43
Wind_Speed_PM                44
Max_WindGust_Direction       16
Max_WindGust_Speed           67
Significant_Rainfall          2
Rainfall_Amount             666
Humidity_AM                 165
Humidity_PM                 166
Atmospheric_Pressure_AM     545
Atmospheric_Pressure_PM     548
CloudCover_Oktas_AM          10
CloudCover_Oktas_PM          10
Burn_Tomorrow                 2
dtype: int64
```

	Minimum_Temperature	Maximum_Temperature	Temperature_AM	\
count	134110.000000	134387.000000	134692.000000	
mean	12.144484	23.321705	16.970687	
std	6.428055	7.214363	6.554124	
min	-8.500000	-4.800000	-7.200000	
25%	7.500000	17.900000	12.200000	
50%	11.900000	22.800000	16.600000	
75%	16.800000	28.500000	21.600000	
max	33.900000	48.100000	40.200000	

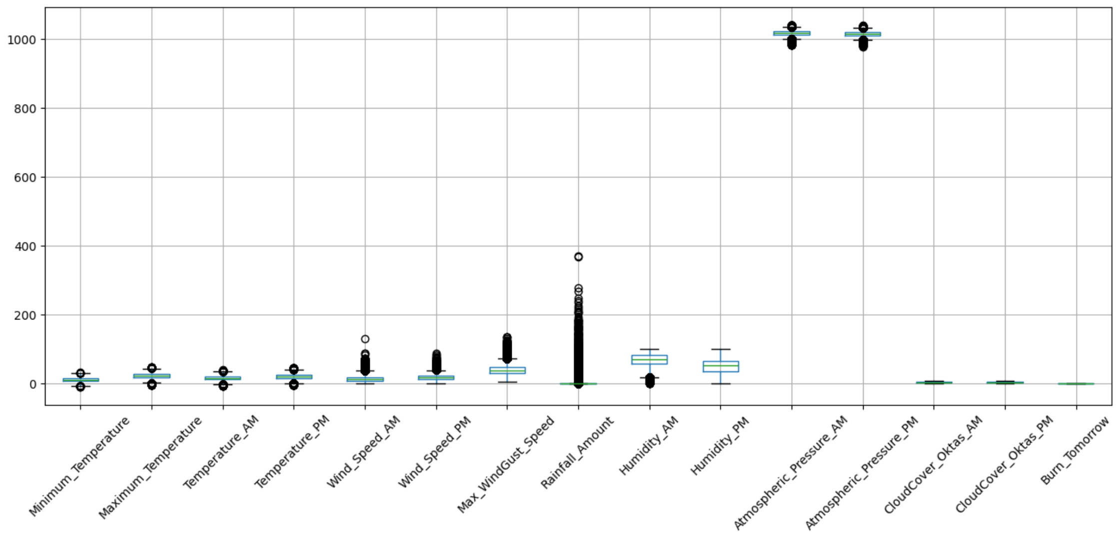
	Temperature_PM	Wind_Speed_AM	Wind_Speed_PM	Max_WindGust_Speed	\
count	134692.000000	133369.000000	132240.000000	125724.000000	
mean	21.792406	13.951293	18.663203	40.126173	
std	7.037912	8.892540	8.836756	13.598228	
min	-5.400000	0.000000	0.000000	6.000000	
25%	16.600000	7.000000	13.000000	31.000000	
50%	21.200000	13.000000	19.000000	39.000000	
75%	26.700000	19.000000	24.000000	48.000000	
max	46.700000	130.000000	87.000000	135.000000	

	Rainfall_Amount	Humidity_AM	Humidity_PM	Atmospheric_Pressure_AM	\
count	133342.000000	134692.000000	134692.000000	134692.000000	
mean	2.344291	68.645683	51.139806	1017.512641	
std	8.472936	19.079176	20.719840	6.775998	
min	0.000000	0.000000	0.000000	982.000000	
25%	0.000000	57.000000	36.000000	1013.300000	
50%	0.000000	70.000000	52.000000	1017.900000	
75%	0.600000	83.000000	65.000000	1021.600000	
max	371.000000	100.000000	100.000000	1041.000000	

	Atmospheric_Pressure_PM	CloudCover_Oktas_AM	CloudCover_Oktas_PM	\
count	134692.000000	134692.000000	134692.000000	
mean	1015.108536	4.517492	4.523015	
std	6.709368	2.375169	2.189380	
min	977.100000	0.000000	0.000000	
25%	1010.800000	3.000000	3.000000	
50%	1015.600000	4.000000	4.000000	
75%	1019.200000	6.000000	6.000000	
max	1039.600000	9.000000	9.000000	

	Burn_Tomorrow
count	134692.000000
mean	0.277648
std	0.447841
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000

```
max          1.000000
```



[6]:

```python
# Extract the month from the 'Date' column
df['Date'] = pd.to_datetime(df['Date'])
df['Month'] = df['Date'].dt.month

# Columns for imputation
numerical_columns_to_impute = ['Minimum_Temperature', 'Maximum_Temperature', 'Wind_Speed_AM', 'Wind_Speed_PM',
                               'Rainfall_Amount', 'Max_WindGust_Speed']
categorical_columns_to_impute = ['Wind_Direction_AM', 'Wind_Direction_PM', 'Max_WindGust_Direction', 'Climate', 'Season',
                                 'Significant_Rainfall']

# Impute numerical columns with monthly median
median_imputer = SimpleImputer(strategy='median')
for col in numerical_columns_to_impute:
    df[col] = df.groupby('Month')[col].transform(lambda x: median_imputer.fit_transform(x.values.reshape(-1, 1)).flatten())

# Impute categorical columns with monthly mode
mode_imputer = SimpleImputer(strategy='most_frequent')
for col in categorical_columns_to_impute:
    df[col] = df.groupby('Month')[col].transform(lambda x: mode_imputer.fit_transform(x.values.reshape(-1, 1)).flatten())

# Drop the 'Month' column not needed anymore
df.drop('Month', axis=1, inplace=True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[6], line 2
      1 # Extract the month from the 'Date' column
----> 2 df['Date'] = pd.to_datetime(df['Date'])
      3 df['Month'] = df['Date'].dt.month
      5 # Columns for imputation

File E:\Python\Lib\site-packages\pandas\core\tools\datetimes.py:1112, in to_datetime(arg, errors, dayfirst, yearfirst, utc, format, exact, unit, infer_datetime_format, origin, cache)
   1110         result = arg.map(cache_array)
   1111     else:
-> 1112         values = convert_listlike(arg._values, format)
   1113         result = arg._constructor(values, index=arg.index, name=arg.name)
   1114 elif isinstance(arg, (ABCDataFrame, abc.MutableMapping)):

File E:\Python\Lib\site-packages\pandas\core\tools\datetimes.py:488, in _convert_listlike_datetimes(arg, format, name, utc, unit, errors, dayfirst, yearfirst, exact)
    486 # `format` could be inferred, or user didn't ask for mixed-format parsing.
    487 if format is not None and format != "mixed":
--> 488     return _array_strptime_with_fallback(arg, name, utc, format, exact, errors)
    490 result, tz_parsed = objects_to_datetime64ns(
    491     arg,
    492     dayfirst=dayfirst,
   (...)
    496     allow_object=True,
    497 )
    499 if tz_parsed is not None:
    500     # We can take a shortcut since the datetime64 numpy array
    501     # is in UTC

File E:\Python\Lib\site-packages\pandas\core\tools\datetimes.py:519, in _array_strptime_with_fallback(arg, name, utc, fmt, exact, errors)
    508 def _array_strptime_with_fallback(
    509     arg,
    510     name,
   (...)
    514     errors: str,
    515 ) -> Index:
    516     """
    517     Call array_strptime, with fallback behavior depending on 'errors'.
    518     """
```

```
--> 519     result, timezones =␣
↪array_strptime(arg, fmt, exact=exact, errors=errors, utc=utc)
    520     if any(tz is not None for tz in timezones):
    521         return _return_parsed_timezone_results(result, timezones, utc,␣
↪name)

File strptime.pyx:534, in pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:355, in pandas._libs.tslibs.strptime.array_strptime()

ValueError: time data "13/07/2013" doesn't match format "%m/%d/%Y", at position␣
↪9. You might want to try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not␣
↪necessarily in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each element␣
↪individually. You might want to use `dayfirst` alongside this.
```

Further Data Preparation for Modelling

[ ]:
```
# List of categorical columns to one-hot encode
categorical_columns = ['City_Name', 'City_State', 'Climate', 'Season',␣
↪'Significant_Rainfall',
                       'Wind_Direction_AM', 'Wind_Direction_PM',␣
↪'Max_WindGust_Direction']

# Apply one-hot encoding to categorical columns
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Print the first few rows of the encoded DataFrame
print(df_encoded.head())
```

[ ]:
```
# List of categorical columns to one-hot encode
categorical_columns = ['City_Name', 'City_State', 'Climate', 'Season',␣
↪'Significant_Rainfall',
                       'Wind_Direction_AM', 'Wind_Direction_PM',␣
↪'Max_WindGust_Direction']

# Apply one-hot encoding to categorical columns
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Print the first few rows of the encoded DataFrame
print(df_encoded.head())
```

[ ]:
```
# Separate features and target variable
X = df_encoded.drop('Burn_Tomorrow', axis=1)
y = df_encoded['Burn_Tomorrow']
```

Class Imbalance Handling

```
[ ]: # Step 1: Feature Scaling
scaler = StandardScaler()
# Exclude the 'Date' column from X
X_scaled = scaler.fit_transform(X.drop('Date', axis=1))

# Import the SMOTE class
from imblearn.over_sampling import SMOTE

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the features and target variables
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Print the shape of the resampled data
print("Shape of X_resampled:", X_resampled.shape)
print("Shape of y_resampled:", y_resampled.shape)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
↪test_size=0.2, random_state=42)

# Print the shape of train and test sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

Feature Selection

```
[ ]: from sklearn.ensemble import RandomForestClassifier

# Assuming X_train and y_train are your training data
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Get feature importances
feature_importances = model.feature_importances_

# Rank features by importance
feature_ranking = sorted(range(len(feature_importances)), key=lambda i:␣
↪feature_importances[i], reverse=True)

# Select top N features (e.g., top 10)
top_n_features = feature_ranking[:10]
```

```
# Create a new dataset with the selected features
X_train_selected = X_train[:, top_n_features]
X_test_selected = X_test[:, top_n_features]

# Print the shape of train_selected and test_selected sets
print("Shape of X_train_selected:", X_train_selected.shape)
print("Shape of X_test_selected:", X_test_selected.shape)
```

[ ]:
```
# List of feature names in the original dataset
feature_names = X.columns.tolist()

# Get the names of the top selected features
selected_feature_names = [feature_names[i] for i in top_n_features]

# Print the names of the selected features
print('selected_feature_names:\n', selected_feature_names)
```

Training and Evaluation

[ ]:
```
from sklearn.metrics import classification_report, accuracy_score

# Create a Random Forest Classifier
random_forest_classifier = RandomForestClassifier(random_state=42)

# Train the Random Forest Classifier on the selected features
random_forest_classifier.fit(X_train_selected, y_train)

# Perform cross-validation on the Decision Tree Classifier
scores = cross_val_score(random_forest_classifier, X_train_selected, y_train,␣
 ↪cv=5)

# Print the cross-validation scores
print("Cross-validation scores:", scores)

# Calculate the mean and standard deviation of the cross-validation scores
mean_score = scores.mean()
std_score = scores.std()

# Print the mean accuracy and standard deviation
print("Mean accuracy:", mean_score)
print("Standard deviation:", std_score)

# Make Predictions
y_pred_rf = random_forest_classifier.predict(X_test_selected)

# Evaluate the Random Forest Classifier
accuracy_rf = accuracy_score(y_test, y_pred_rf)
```

```python
classification_report_rf = classification_report(y_test, y_pred_rf)

# Print the accuracy and classification report for Random Forest
print("Random Forest Classifier:")
print("Accuracy:", accuracy_rf)
print("Classification Report:")
print(classification_report_rf)
```

[ ]:

```python
confusion_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix:")
print(confusion_matrix_rf)
```