Australian Bushfire Detection Using Machine Learning and Neural Networks

[1] Nihal Kumar
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, India
nihal.kumar2017@vitstudent.ac.in

[2] Aditya Kumar
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, India
aditya.kumar2017@vitstudent.ac.in

***Abstract*—Forest fires are increasingly one of the most predominant and alarming disasters in the planet right now and preventing it is very important in order to protect the environment and thousands of animals and plants species that depend on it. The 2019–20 Australian bushfire caused serious uncontrolled fires throughout the summer which burnt millions of hectares of land, destroyed thousands of buildings and killed many people. It has also been estimated to have killed about a billion animals and has bought endangered species on the brink of extinction. Such catastrophic events cannot be allowed to be repeated again. The primary goal of this paper is to improve the efficiency of forest fire detection system of Australia. Data mining and machine learning techniques can help to anticipate and quickly detect fires and take immediate action to minimise the damage. In this paper we try to focus on the implementation of a set of well-known classification algorithms (K-NN and Artificial Neural Networks), which can reduce the existing disadvantages of the fire detection systems. Results from the Kaggle dataset infer that our ANN-MLP algorithm (Multilayer Perceptron) yields better performance by calculating confusion matrix that in turn helps us to calculate performance measure as Detection Rate Accuracy. All predictions and calculations are done with the help of data collected by LANCE FIRMS operated by NASA's Earth Science Data and Information System (ESDIS). The training and testing of the model was done using University of Maryland dataset and was implemented using python.**

***Index Terms*—environment, multilayer perceptron, efficiency, detection rate accuracy, machine learning**

I. Introduction

A wildfire, which is often known as a forest fire, brush fire, or bushfire, is an uncontrolled fire mainly occurring in wildland areas, which is also capable of consuming houses or agricultural lands. Analysing and Identifying the accurate patterns of the causes of these fires is a major problem as physical sensors and fire detectors get damaged in the fires and it is not feasible to place sensors in all the places, therefore the most convenient way of getting the data is through satellite data and imagery because of their relative ease of processing and interpretation. To approach this problem we use Machine Learning and Neural Network algorithms to provide classification techniques. K-Nearest Neighbors classification technique is one of the supervised learning algorithms which has intense application in areas such as pattern recognition, intrusion detection and data mining. On the other hand Multi Layer Perceptrons (MLPs) is comprised of an important class of feed-forward Artificial Neural Networks (ANNs), primarily aimed to replicate learning and generalisation abilities of a person with an attempt to model the functions of biological neural networks. In this paper our aim is to establish the usefulness of the MLP as a pattern classifier compared to the K-Nearest Neighbor (KNN) classifier used as a suboptimal traditional classifier.

II. Related Works

R. Shanmuga priya, K. Vani (2019) [1] a CNN based fire detection model was proposed to spontaneously draw out features, the architecture also achieved high detection rates.

Tahira Islam Trishna, Shimoul Uddin Emon, Romana Rahman Ema (2019) [2] an automatic hepatitis virus recognition system was proposed, which was accomplished by different data mining techniques on Weka. The system made use of naive bayes, K-nearest and Random Forest classifier in WEKA software for evaluation of the result. The application was done on actual data of hepatitis patients, which resulted in effective for diagnosing the hepatitis viruses.

Mary Shermila A, Amrith Basil Bellarmine, Nirmala Santiago (2018) [3] A model was proposed to predict the characteristics of the culprit by using Multilinear Regression, K-Neighbor Classifier and Neural Networks. The resulting model had an accuracy of 0.60 for culprit's age, 0.96 for culprit's sex and 0.97 for culprit's relationship.

Yawen Li, Guangcan Tang, Jiameng Du, Nan Zhou, Yue Zhao, and Tian Wu (2019) [4] For the prediction of fuel consumption of light-duty vehicles, a model was proposed to find the factors that impact the consumption rate of light-duty vehicles in the real world. The model was developed using multilayer perceptron.

Taewon Moon, Seojung Hong, Ha Young Choi, Dae Ho Jung, Se Hong Chang, Jung Eek Son (2019) [5] Using multilayer perceptron and random forest as machine learning algorithms and linear and spline interpolations as statistical

978-1-7281-7223-1/20/$31.00©2020 IEEE

methods, a study of the interpolation accuracy of greenhouse environment data was demonstrated. As a result the statistical models proved to have a high accuracy in predicting the random extraction data and short-term missing data, but the accuracy in predicting long-term interpolation was low. Multilayer perceptron on the other hand produced high accuracy in all experiments.

III. Methodology

A. Dataset

The dataset is extracted from Kaggle, a data science community consisting of repository of large number of domains. The dataset provides physical factors which is brought together by LANCE FIRMS which is controlled by NASA's Earth Science Data and Information System (ESDIS). In the dataset there's 36011 rows and 15 columns who's description is given in TABLE I.

B. WorkFlow Diagram

Fig 1 interprets the methodology for the whole process. Firstly, we split the dataset into training and test set. Using training set, two prosperous classification algorithms that is K-Nearest Neighbors classification and Multilayer Perceptron, was processed to generate the detection models. In the end, with the help of generated classification models, we classified the new and unseen test data and choose the model with the highest accuracy.

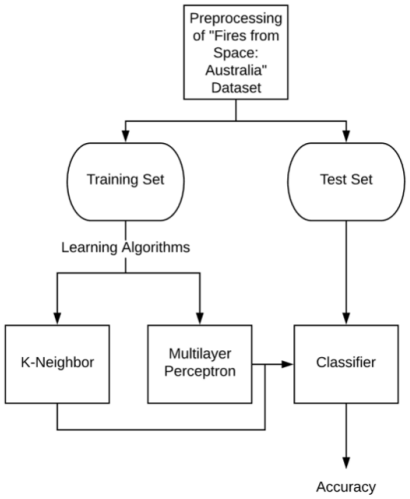

Fig. 1. Workflow diagram

C. Preprocessing

In machine learning, data pre-processing is the step where the data gets converted, transformed or encoded into a state in which machine can easily parse it. Putting it simply, now the features of the data can be easily understood by the algorithm.

The following pre-processing was done:

- Feature Encoding : Feature encoding is essentially carrying out changes on the data in such a way that it can be taken as the input by the algorithm while still maintaining its initial meaning.

- Train / Validation / Test Split : Before deciding the algorithm to be used, it is always recommended to split the dataset into 2 or sometimes 3 parts. Machine Learning algorithms, or any algorithm for that matter, must be trained on the data distribution and then tested and validated, before deploying it to tackle real world data.

- Scaling information using MinMaxScaler : To scale every characteristic to a given variety we use MinMaxScaler. It scales and translates every feature one by one in a way that it is in the given range on the training set.

IV. Implementation

A. Algorithms

1) K-Nearest Neighbor Classification: The basic idea behind K-Nearest Neighbor(KNN) Classifier is that similar objects are near to each other. K-Nearest Neighbor(KNN) is a classification algorithm which is used to classify the target variable with the help of given set of input variables. KNN Algorithm can be applied to various datasets in the domain of finance, healthcare, handwriting detection, and political science. For example, In the domain of finance we can actually predict whether the customer is eligible for loan or not. KNN is the easiest algorithm to implement as compared to other Classification Algorithms. This Algorithm can be used not only for classification problems, but also for the regression-based prediction.

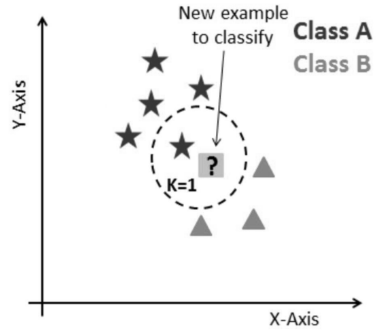

Fig. 2. Graphical Representation of a data

In the given graphical representation of the data as shown in Fig. 2, we can see that there are two classes A and B, which has to be classified using KNN Classifier. Suppose we select P1 point as the data point where we have to predict the class to which it belongs after selecting an appropriate value of K. Similar data points are near to each other and the distance between them will be less as compared to other points. So, to calculate the distance between all the data points we use various distance measures like Euclidean distance, Manhattan distance, Hamming distance or Minkowski distance.

978-1-7281-7223-1/20/$31.00©2020 IEEE

TABLE I
DATASET

No	Name	Type of column	Description
1	Latitude	Numeric	Core of 1 km fire but not accurately the real position of the fire due to the fact that more than one fire can be spotted within 1 km.
2	Longitude	Numeric	Core of 1 km fire but not accurately the real position of the fire due to the fact that more than one fire can be spotted within 1 km.
3	Brightness-Temperature 21	Numeric	Reading of the Channel 21/22 brightness temperature due to the fire pixel measured in Kelvin.
4	Along Scan Pixel Size	Numeric	This algorithm generates 1 km fire pixels but MODIS pixels gets enlarged at the edge of scan. Scan and Track giveback the real pixel size.
5	Along Track pixel size	Numeric	This algorithm generates 1 km fire pixels but MODIS pixels gets enlarged at the edge of scan. Scan and Track giveback the real pixel size.
6	Acquisition Date	Date	Date of the MODIS object obtained.
7	Acquisition Time	Time	Time of the object obtained/ passing of the satellite in Coordinated Universal Time.
8	Satellite	String	A stands for Aqua and T stands for Terra.
9	Instrument	String	Constant value for MODIS.
10	Confidence (0-100%)	Numeric	This value is purposefully calculated to assist the users to measure the grade of a particular hotspot/fire pixels. This is derived from a set of algorithms used in the sensing process. It predicts a range in between 0 to 100% and subsequently assigns any one of the three classes of fire i.e low-confidence fire, nominal-confidence fire, or high-confidence fire.
11	Version	Numeric	It recognises the collection and source of data processing.
12	Brightness temperature 31	Numeric	Channel 31 Brightness-Temperature due to the fire pixel measured in Kelvin.
13	Fire Radiative Power	Numeric	Represents the pixel-consolidated fire radiative power.
14	Day / Night	Character	D stands for Daytime and N stands for Nighttime
15	Type	Numeric	Target Variable

Euclidean Distance Calculation-

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

	Actual;	Predicted
0	2	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
7198	0	0
7199	0	0
7200	0	0
7201	0	0
7202	0	0

Fig. 3. Comparison Between Actual Value and Predicted Value

We can compare actual data and predicted results of K-Nearest Neighbor Classification in Fig 3.

Confusion Matrix: Fig. 4 shows the confusion matrix of K-Nearest Neighbor Classification.

```
In [52]: TP=confusion[1,1]
         TN = confusion[0, 0]
         FP = confusion[0, 1]
         FN = confusion[1, 0]
         confusion

Out[52]: array([[7122,   11,    0],
                [  55,   11,    0],
                [   4,    0,    0]], dtype=int64)
```

Fig. 4. Confusion matrix of K-Nearest Neighbor Classification.

Classification Error: Fig 5 shows the Classification Error of K-Nearest Neighbor Classification.

```
In [54]: classification_error=(FP+FN)/float(TP+TN+FP+FN)
         classification_error

Out[54]: 0.009167939991665508
```

Fig. 5. Classification Error of K-Nearest Neighbor Classification.

Sensitivity: Fig 6 shows the Sensitivity of K-Nearest Neighbor Classification.

```
In [56]: sensitivity=TP/float(FN + TP)
         sensitivity

Out[56]: 0.16666666666666666
```

Fig. 6. Sensitivity of K-Nearest Neighbor Classification.

Specificity: Fig 7 shows the Specificity of K-Nearest Neighbor Classification.

978-1-7281-7223-1/20/$31.00©2020 IEEE

```
In [58]: specificity = TN / (TN + FP)
         specificity
         #our classifier is highly specific and not sensitive

Out[58]: 0.9984578718631711
```

Fig. 7. Specificity of K-Nearest Neighbor Classification.

False Positive Rate: Fig 8 shows the False Positive Rate of K-Nearest Neighbor Classification.

```
false_positive_rate = FP / float(TN + FP)
false_positive_rate

Out[61]: 0.0015421281368288237
```

Fig. 8. False Positive Rate of K-Nearest Neighbor Classification.

Precision: Fig 9 shows the Precision of K-Nearest Neighbor Classification.

```
precision = TP / float(TP + FP)
precision

Out[63]: 0.5
```

Fig. 9. Precision of K-Nearest Neighbor Classification.

Adjusting the Values of Classification: It is very important to normalize the values so that all different values come in the range of 0 to 1 and to make the implementation more efficient, Fig 10 shows the normalized values and Fig 11 shows the histogram of predicted possibilities.

```
In [23]: # print the first 10 predicted responses
         # 1D array (vector) of binary values (0, 1)
         logreg.predict(X_test)[0:10]

Out[23]: array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1])

In [24]: # print the first 10 predicted probabilities of class membership
         logreg.predict_proba(X_test)[0:10]

Out[24]: array([[ 0.63247571,  0.36752429],
                [ 0.71643656,  0.28356344],
                [ 0.71104114,  0.28895886],
                [ 0.5858938 ,  0.4141062 ],
                [ 0.84103973,  0.15896027],
                [ 0.82934844,  0.17065156],
                [ 0.50110974,  0.49889026],
                [ 0.48658459,  0.51341541],
                [ 0.72321388,  0.27678612],
                [ 0.32810562,  0.67189438]])

In [69]: # print the first 10 predicted probabilities for class 1
         learner.predict_proba(x_test)[0:10, 1]

Out[69]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

In [70]: # store the predicted probabilities for class 1
         y_pred_prob = learner.predict_proba(x_test)[:, 1]
```

Fig. 10. Adjusting the Values of Classification

2) Multi-layer Perceptron (MLP): A multi-layer perceptron is an Artificial Neural Network algorithm [6] . In this neural network, we have basically three major layers. The first layer is the input layer and the middle layer comprises of various inner layers which can vary according to the model needs and the last layer is our output layer. Each layer can have several neurons. But the number of neurons in the last output layer is exactly same as the number of classes which we need to predict. So, initially each neuron in the layer is assigned with some random weights and with the help of given output the model tries to figure out the error difference between the actual values and the predicted values, using a function or a method known as Activation function , the most commonly used activation function us the RELU function. So, after few iterations the model assigns appropriate weight to each neuron which finally perform their operations and tries to predict the output with the given set of test data. This Classification algorithm is one of the most efficient classifier which can give the highest accuracy of the model. Multilayer perceptrons are usually applied to supervised learning [7] problems and prediction models. This model basically tries to learn the correlation between the input and the output layer. Fig 11 shows the comparison between actual data and the predicted values using Multi-layer Perceptron.

	Actual;	Predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
25203	0	0
25204	0	0
25205	0	0
25206	0	0
25207	0	0

25208 rows × 2 columns

Fig. 11. Comparison Between Actual Value and Predicted Value

Confusion Matrix: Fig 12 shows the confusion matrix of Multi-layer Perceptron.

```
In [40]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import confusion_matrix
         confusion=confusion_matrix(ya,yp)
         TP=confusion[1,1]
         TN = confusion[0, 0]
         FP = confusion[0, 1]
         FN = confusion[1, 0]
         confusion

Out[40]: array([[25199,     0],
                [    9,     0]], dtype=int64)
```

Fig. 12. Confusion Matrix of Multi-layer Perceptron.

Classification Error: Fig 13 shows the Classification Error of Multi-layer Perceptron.

978-1-7281-7223-1/20/$31.00©2020 IEEE

```
In [41]: classification_error=(FP+FN)/float(TP+TN+FP+FN)
         classification_error
Out[41]: 0.00035702951443986037
```

Fig. 13. Classification Error of Multi-layer Perceptron.

Sensitivity: Fig 14 shows the Sensitivity of Multi-layer Perceptron.

```
In [82]: sensitivity=TP/float(FN + TP)
         sensitivity
Out[82]: 0.0
```

Fig. 14. Sensitivity of Multi-layer Perceptron.

Specificity: Fig 15 shows the Specificity of Multi-layer Perceptron.

```
In [83]: specificity = TN / (TN + FP)
         specificity
Out[83]: 1.0
```

Fig. 15. Specificity of Multi-layer Perceptron.

False Positive Rate: Fig 16 shows the False Positive Rate of Multi-layer Perceptron.

```
In [84]: false_positive_rate = FP / float(TN + FP)
         false_positive_rate
Out[84]: 0.0
```

Fig. 16. False Positive Rate of Multi-layer Perceptron.

Precision: Fig 17 shows the Precision of Multi-layer Perceptron.

```
In [85]: precision = TP / float(TP + FP)
         precision

         C:\Users\Aditya\Anaconda3\lib\site
           """Entry point for launching an

Out[85]: nan
```

Fig. 17. Precision of Multi-layer Perceptron.

Adjusting the Values of Classification: Fig 18 shows the normalized values.

```
In [88]: # print the first 10 predicted responses
         # 1D array (vector) of binary values (0, 1)
         learner.predict(x_test)[0:10]
         # print the first 10 predicted probabilities of class membership
         learner.predict_proba(x_test)[0:10]
Out[88]: array([[9.99600616e-01, 3.99383743e-04],
                [9.99600616e-01, 3.99383743e-04],
                [9.99974986e-01, 2.50137614e-05],
                [9.99974986e-01, 2.50137614e-05],
                [9.99999852e-01, 1.47590868e-07],
                [9.99600616e-01, 3.99383743e-04],
                [9.99997158e-01, 2.84190904e-06],
                [9.99600616e-01, 3.99383743e-04],
                [9.99814698e-01, 1.85302216e-04],
                [9.99600616e-01, 3.99383743e-04]])

In [89]: # print the first 10 predicted probabilities for class 1
         learner.predict_proba(x_test)[0:10, 1]
Out[89]: array([3.99383743e-04, 3.99383743e-04, 2.50137614e-05, 2.50137614e-05,
                1.47590868e-07, 3.99383743e-04, 2.84190904e-06, 3.99383743e-04,
                1.85302216e-04, 3.99383743e-04])

In [91]: # store the predicted probabilities for class 1
         y_pred_prob = learner.predict_proba(x_test)[:, 1]
         y_pred_prob
Out[91]: array([3.99383743e-04, 3.99383743e-04, 2.50137614e-05, ...,
                2.50137614e-05, 3.99383743e-04, 7.57337267e-05])
```

Fig. 18. Adjusting the Values of Classification

3) Performance Metrics of K-Neighbor Classifier:

- R2-score: Fig 19 shows the R2 value of K-Neighbor Classifier.

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

```
In [49]: from sklearn.metrics import r2_score

In [50]: r2_score(ya, yp)
Out[50]: -0.009688977106427243
```

Fig. 19. R2 value of K-Neighbor Classifier

- Accuracy Score: Fig 20 shows the accuracy score of K-Neighbor Classifier.

$$\text{Accuracy score } = \frac{1}{N} \sum_{i=0}^{N-1} 1\left(\hat{y}_i = y_i\right)$$

```
In [44]: from sklearn.metrics import accuracy_score

In [45]: acc=accuracy_score(ya,yp)*100

In [46]: acc
Out[46]: 99.02818270165209
```

Fig. 20. Accuracy of K-Neighbor Classifier

- F1 Score: Fig 21 shows the F1 Score of K-Neighbor Classifier.

$$\mathbf{F1\_Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{precision} + \text{Recall}}$$

978-1-7281-7223-1/20/$31.00©2020 IEEE

```
In [69]: from sklearn.metrics import f1_score

In [70]: f1_score(ya,yp,average='macro')

         C:\Users\Aditya\Anaconda3\lib\site-pa
         ned and being set to 0.0 in labels wi
           'precision', 'predicted', average,

Out[70]: 0.4150365609426669
```

Fig. 21. F1 Score of K-Neighbor Classifier

- Jaccard Score: Fig 22 shows the Jaccard Score of K-Neighbor Classifier.

$$\mathrm{J} = \frac{y_i \cap \hat{y}_i}{y_i \cup \hat{y}_i}$$

```
In [71]: from sklearn.metrics import jaccard_score

In [72]: jaccard_score(ya, yp, average=None)

Out[72]: array([0.99026696, 0.14285714, 0.        ])
```

Fig. 22. Jaccard Score of K-Neighbor Classifier

4) Performance Metrics of Multi-layer Perceptron:

- R2-score: Fig 23 shows the R2 value of Multi-layer Perceptron.

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

```
In [43]: from sklearn.metrics import r2_score

In [45]: r2_score(ya, yp)

Out[45]: -0.00035715703004091104
```

Fig. 23. R2 value of Multi-layer Perceptron

- Accuracy Score: Fig 24 shows the accuracy score of Multi-layer Perceptron.

$$\text{Accuracy score} = \frac{1}{N} \sum_{i=0}^{N-1} 1\left(\hat{y}_i = y_i\right)$$

```
In [38]: from sklearn.metrics import accuracy_score

In [39]: acc=accuracy_score(ya,yp)*100

In [40]: acc

Out[40]: 99.96429704855602
```

Fig. 24. Accuracy of Multi-layer Perceptron

- F1 Score: Fig 25 shows the F1 Score of Multi-layer Perceptron.

$$\mathbf{F1\_Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{precision} + \text{Recall}}$$

```
In [92]: from sklearn.metrics import f1_score

In [94]: f1_score(ya,yp,average='macro')

         C:\Users\Aditya\Anaconda3\lib\site-pa
         ned and being set to 0.0 in labels wi
           'precision', 'predicted', average,

Out[94]: 0.49991072668478587
```

Fig. 25. F1 Score of Multi-layer Perceptron

- Jaccard Score: Fig 26 shows the Jaccard Score of Multi-layer Perceptron.

$$\mathrm{J} = \frac{y_i \cap \hat{y}_i}{y_i \cup \hat{y}_i}$$

```
In [95]: from sklearn.metrics import jaccard_score

In [97]: jaccard_score(ya, yp, average=None)

Out[97]: array([0.99964297, 0.        ])
```

Fig. 26. Jaccard Score of Multi-layer Perceptron

5) Comparisons using Bar plots:

- R2 Score

```
In [91]: #compare r2_score()
         name=["KNN","MLP"]
         res=[r1,r2]

In [92]: plot=plt.bar(name,res)
```

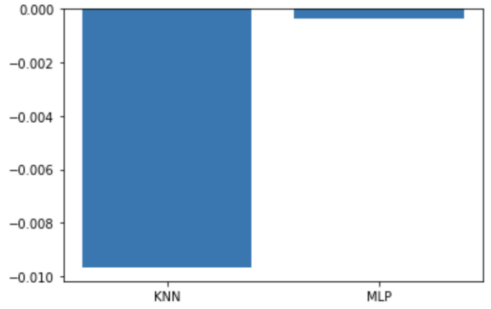

Fig. 27. R2 Scores of KNN and MLP

- Accuracy Score

978-1-7281-7223-1/20/$31.00©2020 IEEE

```
In [90]: plot=plt.bar(names,accuracy)
```

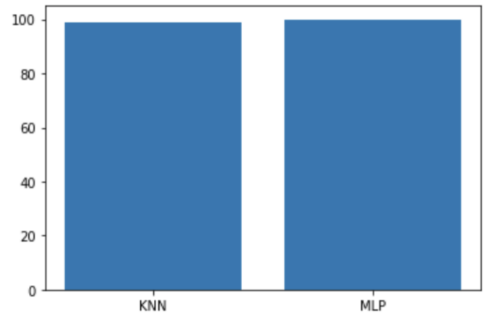

Fig. 28. Accuracy Scores of KNN and MLP

- F1 Score

```
In [100]: plot=plt.bar(n,l)
```

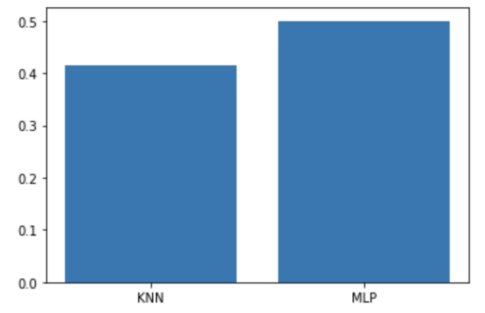

Fig. 29. F1 Scores of KNN and MLP

- Jaccard Score

```
In [63]: #Jaccard Graph Comparision
         j1=0.37 #KNN
         j2=0.49 #MPL
         n=["KNN","MLP"]
         l=[j1,j2]
         plots=plt.bar(n,l)
```

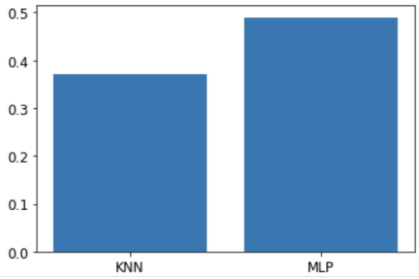

Fig. 30. Jaccard Scores of KNN and MLP

V. Conclusion and Future Work

By our model, we tired to get insights into the data that will be helpful in making effective decisions for the detection of wildfire. In this paper, we have generated fire detection and classification models using multilayer perceptron algorithm and K-Nearest Neighbor algorithm and compared them with each other. The Multilayer perceptron algorithm has the highest accuracy with 99.96%. The accuracy of this algorithm is higher than the accuracy of K-Nearest Neighbor algorithm . Future work can include a comparison of the other more up-to-date algorithms obtaining high accuracy as well as less complexity.

Acknowledgments

We would like to express our sincere gratitude towards the Department of Computer Science and Engineering, Vellore Institute of Technology as without their proactive support and guidance this paper would not have been possible.

References

[1] R. Shanmuga priya and K. Vani, *Deep Learning Based Forest Fire Classification and Detection in Satellite Images*, 11 th. International Conference on Advanced Computing (ICoAC), IEEE, 2019.

[2] Tahira Islam Trishna, Shimoul Uddin Emon and Romana Rahman Ema*Detection of Hepatitis (A, B, C and E) Viruses Based on Random Forest, K-nearest and Naïve Bayes Classifier*, 10 th. International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, 2019.

[3] Mary Shermila A, Amrith Basil Bellarmine and Nirmala Santiago *Crime Data Analysis and Prediction of Perpetrator Identity using Machine Learning Approach*, 2 nd. International Conference on Trends in Electronics and Informatics (ICOEI), IEEE, 2018.

[4] Yawen Li, Guangcan Tang, Jiameng Du, Nan Zhou, Yue Zhao and Tian*Multilayer Perceptron Method to Estimate Real-World Fuel Consumption Rate of Light Duty Vehicles*, Special Section on Ai-driven Big Data Processing: Theory, Methodology, And Applications, IEEE, 2019.

[5] Taewon Moon, Seojung Hong, Ha Young Choi, Dae Ho Jung, Se Hong Chang and Jung Eek Son*Interpolation of greenhouse environment data using multilayer perceptron*, Computers and Electronics in Agriculture 166, 105023, 2019.

[6] Roshini TV, Ravi RV, Reema Mathew A, Kadan AB, Subbian PS. *Automatic Diagnosis of Diabetic Retinopathy with the Aid of Adaptive Average Filtering with Optimized Deep Convolutional Neural Network*. International Journal of Imaging Systems and Technology, Wiley, 2020.

[7] Muthusamy SP, Raju J, Ashwin M, Ravi RV, Prabaker ML, Subramaniam K. *Synergic Deep Learning Based Preoperative Metric Prediction and Patient Oriented Payment Model for Total Hip Arthroplasty*. Journal of Ambient Intelligence and Humanized Computing, Springer, 2020.

978-1-7281-7223-1/20/$31.00©2020 IEEE