Exploring Wine Classification with K-Nearest Neighbors & Random Forest: A Comprehensive Analysis

Gogineni Sai Rohith
Dept of Computer Science Engineering
SRM UNIVERSITY AP, India
sairohith_gogineni@srmap.edu.in

Chaitanya Sai Nutakki
Dept of Computer Science Engineering
SRM UNIVERSITY AP, India
sainutakki_chaitanya@srmap.edu.in

Saleti Sumalatha
Dept of Computer Science Engineering
SRM UNIVERSITY AP, India
sumalatha.s@srmap.edu.in

Atluri Jithin Chowdary
Dept of Computer Science Engineering
SRM UNIVERSITY AP, India
jithinchowdary_atluri@srmap.edu.in

Ritesh Jadhav
Dept of Computer Science Engineering
SRM UNIVERSITY AP, India
ritesh_1@srmap.edu.in

Uday sai Chaganti
Dept of Computer Science Engineering
SRM UNIVERSITY AP, India
udaysai_chaganti@srmap.edu.in

Abstract

This project focuses on exploring the Wine dataset, a classic dataset widely used in the field of machine learning. The primary objective is to conduct a thorough exploratory analysis to gain insights into the dataset's characteristics and distribution. The dataset contains various attributes related to wine samples, such as chemical composition and origin. Although the dataset doesn't inherently pose a specific problem, the goal is to leverage data analysis techniques to extract meaningful insights and knowledge. Subsequently, the K-Nearest Neighbors (KNN) algorithm will be implemented for classification tasks based on the dataset's features. By applying KNN, we aim to classify wine samples into predefined classes based on their similarity to neighboring samples. This project serves as an educational exercise to understand data exploration, feature analysis, and the application of the KNN algorithm in a simple yet effective manner.

Keywords **- Wine dataset, Exploratory analysis, K-Nearest Neighbors (KNN), Classification, Data insights**

I. Introduction

In the realm of viticulture and oenology, the pursuit of quality and authenticity is paramount. Central to this endeavor is the ability to accurately discern the provenance of wine samples based on their chemical composition. This task, while complex, holds significant implications for both industry practitioners and consumers alike.

In this research endeavor, we undertake a comprehensive exploration of the Wine dataset, a classic repository of viticultural data widely utilized in the field of machine learning. Our objective is twofold: to conduct an in-depth exploratory analysis of the dataset and to implement the K-Nearest Neighbors (KNN) algorithm for wine classification.

The Wine dataset, renowned for its simplicity and accessibility, serves as an ideal substrate for our analytical pursuits. Through rigorous examination and application of data analysis techniques, we endeavor to extract meaningful insights and knowledge pertinent to the classification of wine samples based on their chemical attributes.

At the heart of our inquiry lies the imperative to address a pertinent challenge within the wine industry – the accurate identification of wine origins. By leveraging the KNN algorithm, renowned for its efficacy in pattern recognition tasks, we aspire to develop a robust classification model capable of discerning the geographical provenance of wine samples with a high degree of accuracy.

Against the backdrop of this research, we pose the following problem statement: To develop a classification model that can reliably identify the origin of a wine sample based on its chemical composition. Through systematic investigation and methodological rigor, we endeavor to contribute to the advancement of both theoretical understanding and practical application within the domain of wine classification.

As we embark on this scholarly endeavor, we acknowledge the rich tapestry of knowledge and expertise that precedes us, drawing inspiration from the interdisciplinary intersection of viticulture, data science, and machine learning. It is with a commitment to scholarly rigor and intellectual curiosity that we undertake this exploration, aspiring to contribute meaningfully to the body of knowledge within our respective fields of inquiry.

II. Methodology

A. Data Description

The Wine dataset encompasses 14 features, including one target variable, and comprises 178 distinct labels. Each feature within the dataset is numeric, with no instances of missing values or duplicates. The dataset serves as a reliable foundation for exploratory analysis and classification tasks. As seen below it showcases that our dataset is clean.

```
Out[16]: alcohol                         float64
         malic_acid                      float64
         ash                             float64
         alcalinity_of_ash               float64
         magnesium                       float64
         total_phenols                   float64
         flavanoids                      float64
         nonflavanoid_phenols            float64
         proanthocyanins                 float64
         color_intensity                 float64
         hue                             float64
         od280/od315_of_diluted_wines    float64
         proline                         float64
         class                             int32
         dtype: object
```

The dataset originates from a chemical analysis of wines cultivated in the same region in Italy, yet derived from three distinct cultivars. It offers insights into the quantities of 13 constituents present in each of these wine varieties. The attributes include:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

It's worth noting that the first attribute serves as the class identifier (target), categorized into three classes (1-3).

The dataset, utilized extensively for evaluating various classifiers, presents a well-posed classification problem with structured class distributions. While not particularly challenging, it serves as an ideal candidate for initial testing of new classifiers.

B. Exploratory Data Analysis

Data Overview

.head() is a method used to display the first few rows of a DataFrame or a Series.

```
df_wine.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82

Data Descriptive

you can use df.describe() to get statistics such as count, mean, standard deviation, minimum, maximum, and quartiles for each numerical column in the DataFrame.

```
In [19]: df_wine.describe(include="all")
```

Out[19]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthoc
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.(
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.5
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.5
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.4
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.2
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.5
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.5
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.5

Check for duplicated rows

```
In [21]: df_wine.duplicated().any()

Out[21]: False
```

Verify unique values

```
In [22]: df_wine.apply(lambda x: len(x.unique()))

Out[22]: alcohol                         126
         malic_acid                      133
         ash                              79
         alcalinity_of_ash                63
         magnesium                        53
         total_phenols                    97
         flavanoids                      132
         nonflavanoid_phenols             39
         proanthocyanins                 101
         color_intensity                 132
         hue                              78
         od280/od315_of_diluted_wines    122
         proline                         121
         class                             3
         dtype: int64
```

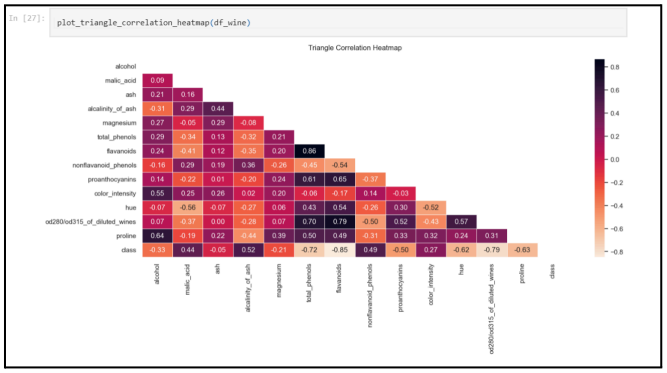
Check target count

```
In [23]: df_wine['class'].value_counts()

Out[23]: class
         1    71
         0    59
         2    48
         Name: count, dtype: int64
```

Features values count

```
In [24]: for feature in df_wine.columns:
             print('-------------------------')
             print(f'Series: {feature}')
             print('-------------------------')
             print(f'{df_wine[feature].value_counts()}\n')

-------------------------
Series: alcohol
-------------------------
alcohol
13.05    6
12.37    6
12.08    5
12.29    4
12.42    3
        ..
13.72    1
13.29    1
13.74    1
13.77    1
14.13    1
Name: count, Length: 126, dtype: int64
```

Check Correlation



Outliers

Outliers points that significantly differ from other observations in the dataset**.** df.outliers specifically for identifying outliers.

In [35]: df_outliers

Out[35]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_int
25	13.05	2.05	3.22	25.0	124.0	2.63	2.68	0.47	1.92	
59	12.37	0.94	1.36	10.6	88.0	1.98	0.57	0.28	0.42	
69	12.21	1.19	1.75	16.8	151.0	1.85	1.28	0.14	2.50	
73	12.99	1.67	2.60	30.0	139.0	3.30	2.89	0.21	1.96	
78	12.33	0.99	1.95	14.8	136.0	1.90	1.85	0.35	2.76	
95	12.47	1.52	2.20	19.0	162.0	2.50	2.27	0.32	3.28	
110	11.46	3.74	1.82	19.5	107.0	3.18	2.58	0.24	3.58	
115	11.03	1.51	2.20	21.5	85.0	2.46	2.17	0.52	2.01	
121	11.56	2.05	3.23	28.5	119.0	3.18	5.08	0.47	1.87	
123	13.05	5.80	2.13	21.5	86.0	2.62	2.65	0.30	2.01	
127	11.79	2.13	2.78	28.5	92.0	2.13	2.24	0.58	1.76	
137	12.53	5.51	2.64	25.0	96.0	1.79	0.60	0.63	1.10	
151	12.79	2.67	2.48	22.0	112.0	1.48	1.36	0.24	1.26	
158	14.34	1.68	2.70	25.0	98.0	2.80	1.31	0.53	2.70	
159	13.48	1.67	2.64	22.5	89.0	2.60	1.10	0.52	2.29	
166	13.45	3.70	2.60	23.0	111.0	1.70	0.92	0.43	1.46	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	

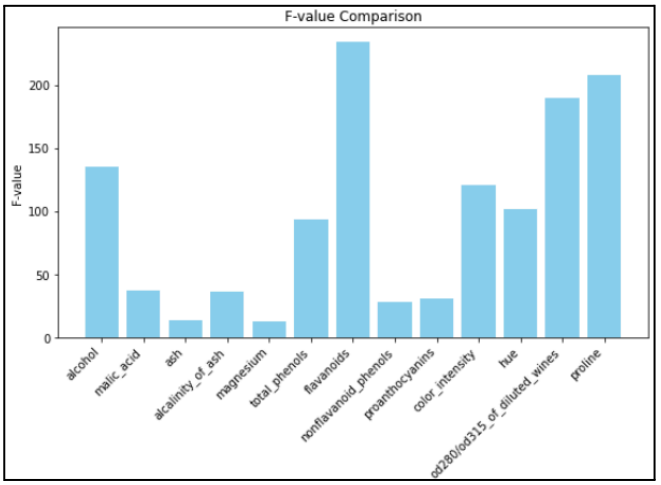
C. Pre-Processing

ANOVA F-value (Feature Selection)

Filter methods involve selecting features independently of machine learning algorithms based on specific criteria, such as statistical scores and variances. These methods offer several advantages, including their independence from machine learning algorithms, making them compatible with any model, and their efficiency in computation time. However, they lack consideration of feature relationships, making them suitable primarily as preprocessing steps in feature selection pipelines.

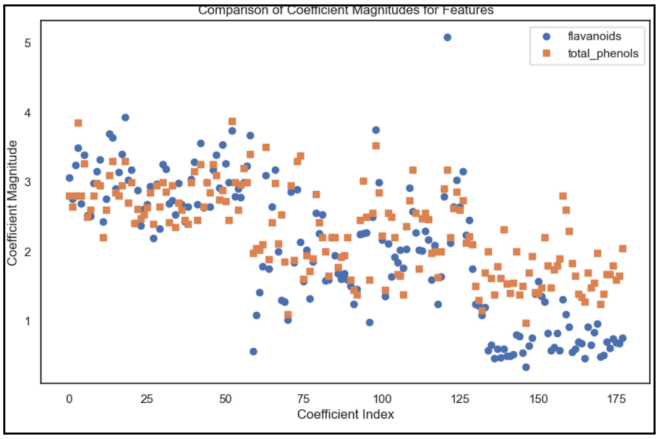
Among the filter selection methods discussed, the ANOVA F-value method estimates the linearity between input and output features. A high F-value indicates a strong linear relationship, while a low F-value suggests a weak one. However, this method is limited to capturing only linear relationships, unable to detect non-linear relationships.

To calculate ANOVA F-values using Scikit-learn, we utilize the `f_classif` function, which is suitable for classification tasks like those presented in the Iris dataset. This function evaluates the F-value between input and output features to inform feature selection.



Plot Coefficient Magnitude Comparison

To plot a comparison of coefficient magnitudes before and after preprocessing in a machine learning pipeline

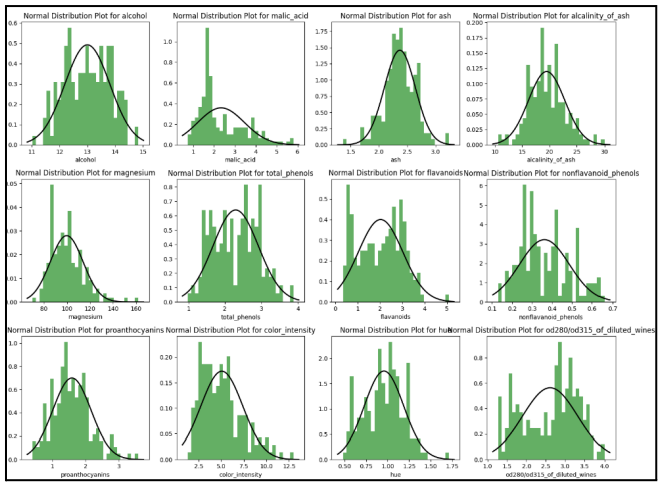


Plot Normal Distribution Comparison

A normal distribution, or Gaussian distribution, is symmetric around its mean, with observations clustering near the mean and tapering off towards the tails. The function `plot_normal_distribution_comparison` compares data column distributions to normal distributions.

1. It creates a grid of subplots, each displaying a histogram of a data column.
2. Overlaid on each histogram is a normal distribution curve calculated from the column's mean and standard deviation.

3. This comparison helps visualize how closely each data column's distribution resembles a normal distribution.



Flavanoids:
Notable correlations emerge between features, with Total Phenols displaying the highest correlation of 0.86, followed by OD280/OD315 of diluted wines at 0.79. Conversely, the Class exhibits the lowest correlation at -0.85.

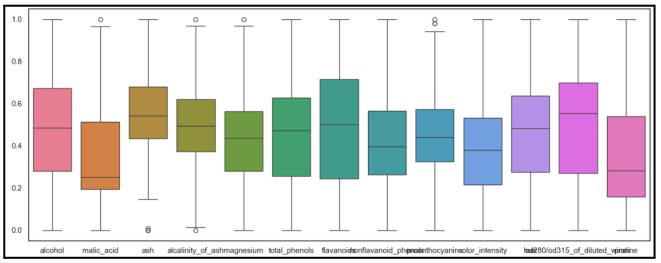
Data Separation:
Proline demonstrates the most distinct separation among features, particularly accentuated for class = 0. Additionally, Color_intensity displays notable separation, especially when combined with Flavanoids and Total Phenols.

Transformation:
The decision to transform the data arises from the sensitivity of the KNN algorithm to varying scales and outliers, particularly in the context of Euclidean distance calculations. The original dataset presents disparate scales, ranging from a minimum of 0.13 in Nonflavanoid Phenols to a maximum of 1680.00 in Proline. Moreover, seven features initially exhibit outliers, including alcalinity_of_ash, magnesium, color_intensity, malic_acid, ash, proanthocyanins, and hue.

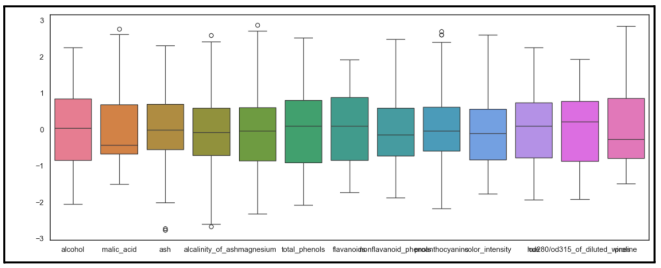
III. Transformation and Cleaning

Normalization and standardization are preprocessing techniques commonly used in machine learning, including KNN (K-Nearest Neighbors) classification, to improve the performance of the algorithms.

Normalization:
It is the process of scaling numeric features to a uniform range. This typically involves scaling the features so that they fall within a range of [0, 1].
In the context of KNN, normalization ensures that all features contribute equally to the distance computations between data points. Without normalization, features with larger scales may dominate the distance calculation, leading to biased results.
By bringing all features to a similar scale, normalization prevents features with larger numeric ranges from outweighing those with smaller ranges when computing distances between data points.



Standardization:
Standardization (also known as z-score normalization) involves transforming the features such that they have a mean of 0 and a standard deviation of 1.
Similar to normalization, standardization helps in making features comparable by scaling them to a standard scale.
In KNN, standardization can be useful when the features have different units or scales. It ensures that the features are centered around zero with a comparable spread, which can improve the performance of distance-based algorithms like KNN.
Standardization is less sensitive to outliers compared to normalization, as it centers the data around the mean.



IV. Results

K-Nearest Neighbors (KNN) is a popular and intuitive algorithm used for both classification and regression tasks in machine learning. It operates on the principle that similar data points tend to belong to the same class or have similar output values. Here's a closer look at some key parameters of the KNeighborsClassifier, a classifier implementation of KNN in sci-kit-learn:

n_neighbors: This parameter specifies the number of neighbors to consider when making predictions. A higher value typically results in a smoother decision boundary but may increase computational complexity.

weights: Determines the weight given to neighboring samples during prediction. The default value, 'uniform', assigns equal weight to each neighbor. Alternatively, setting it to 'distance' gives higher weight to closer neighbors, which can be beneficial if the dataset has varying densities or noisy samples.

metric: Specifies the distance metric used to calculate the distance between data points. The default is 'minkowski', which generalizes both the Euclidean and Manhattan distances. Other common choices include 'euclidean', 'manhattan', 'cosine', and 'mahalanobis', among others.

p: This parameter is relevant when using the Minkowski distance metric. It determines the power parameter for the Minkowski metric. When p=2 (default), it is equivalent to the Euclidean distance; when p=1, it is equivalent to the Manhattan distance.

n_jobs: Determines the number of parallel jobs to run during the neighbor search process. Specifying -1 will use all available CPU cores. This parameter can significantly speed up computation, especially for large datasets.

Normalization Results:

```
---------------------------------
Train Confusion Matrix | Normalized
---------------------------------
Prediction   0   1   2  All
Real
0           44   0   0   44
1            2  42   2   46
2            0   0  30   30
All         46  42  32  120
---------------------------------
Train Score: 0.9666666666666667
```

Random Forest:

The Random Forest is a versatile machine learning model widely used for both classification and regression tasks. It is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random Forest Result:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

def train_and_evaluate_random_forest(scale_name, X, y, n_estimators=100, random_state=37):
    rf = RandomForestClassifier(n_estimators=n_estimators, random_state=random_state)

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=random_state)


    rf.fit(X_train, y_train)


    y_pred = rf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    print(f'Accuracy of Random Forest classifier ({scale_name}): {accuracy:.2f}')


X = df_wine.drop(columns=['class'])
y = df_wine['class']

train_and_evaluate_random_forest('Original', X, y)
```

```
Accuracy of Random Forest classifier (Original): 0.96
```

```
---------------------------------
Test Confusion Matrix | Normalized
---------------------------------
Prediction   0   1   2  All
Real
0           14   0   0   14
1            0  15   0   15
2            0   0  12   12
All         14  15  12   41
Test Score: 1.0
```

```
-----------------------------------------------------
Test Classification Metrics | Normalized
-----------------------------------------------------
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        14
     class_1       1.00      1.00      1.00        15
     class_2       1.00      1.00      1.00        12

    accuracy                           1.00        41
   macro avg       1.00      1.00      1.00        41
weighted avg       1.00      1.00      1.00        41
```

Standardized Results:

```
---------------------------------
Train Confusion Matrix | Standardized
---------------------------------
Prediction   0   1   2  All
Real
0           44   0   0   44
1            2  41   3   46
2            0   0  30   30
All         46  41  33  120
---------------------------------
Train Score: 0.9583333333333334
```

```
---------------------------------
Test Confusion Matrix | Standardized
---------------------------------
Prediction   0   1   2  All
Real
0           14   0   0   14
1            0  15   0   15
2            0   1  11   12
All         14  16  11   41
Test Score: 0.975609756097561
```

```
-----------------------------------------------------
Test Classification Metrics | Standardized
-----------------------------------------------------
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        14
     class_1       0.94      1.00      0.97        15
     class_2       1.00      0.92      0.96        12

    accuracy                           0.98        41
   macro avg       0.98      0.97      0.97        41
weighted avg       0.98      0.98      0.98        41
```

Final Results:

	Scale	Train Score	Test Score
0	Normalized	0.966667	1.00000
1	Standardized	0.958333	0.97561

V. Conclusion

Both normalization and standardization resulted in test and training set accuracies above 95%, suggesting various factors contributing to this outcome:

1. Overfitting:
Achieving 100% accuracy on the test set may indicate overfitting. When a model fits the training data too closely, it struggles to generalize to new data, memorizing patterns instead of learning from them.

2. Size of the Test Set:
- The small size of the test set could contribute to the high accuracy observed. A small test set may inadvertently align well with the nearest neighbors in KNN, leading to seemingly optimal performance without truly reflecting the model's capability.

3. Good Separability:
- KNN performs well when data in the test set are easily separable. Clear distinctions between groups and well-spread test examples facilitate accurate predictions as KNN relies on proximity for classification.

4. Proper Pre-processing:
- Effective data pre-processing, including normalization, outlier treatment, and appropriate preparation of training and test data, significantly impacts model performance. Ensuring proper data treatment enhances the model's ability to learn and generalize from the data.

5. Appropriate Choice of Hyperparameters:
- Hyperparameters like the number of neighbors (k) play a crucial role in KNN's performance. Selecting suitable hyperparameters is essential, as they can greatly influence the model's behavior and ability to generalize effectively.

Considering these factors collectively, achieving high accuracy in both training and test sets reflects not only the effectiveness of pre-processing techniques but also the interplay of various factors influencing KNN's performance.

IX. References

Aich S., Al-Absi A.A., Hui K.L., Lee J.T., Sain M.

A classification approach with different feature sets to predict the quality of different types of wine using machine learning techniques

International Conference on Advanced Communication Technology, ICACT, 2018-February (2018), pp. 139-143, 10.23919/ICACT.2018.8323674

View in ScopusGoogle ScholarAipperspach et al., 2020

Aipperspach A., Hammond J., Hatterman-Valenti H.

Utilizing pruning and leaf removal to optimize ripening of vitis riparia-based ‘frontenac gris’ and ‘marquette’ wine grapes in the northern great plains

Horticulturae, 6 (1) (2020), p. 18, 10.3390/horticulturae6010018

View in ScopusGoogle ScholarAmpomah et al., 2020

Ampomah E.K., Qin Z., Nyame G.

Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement

Information, 11 (6) (2020), p. 332, 10.3390/INFO11060332

11(6) 332

View in Scopus

Google ScholarArauzo-Azofra et al., 2011

Arauzo-Azofra A., Aznarte J.L., Benítez J.M.

Empirical study of feature selection methods based on individual feature evaluation for classification problems

Expert Systems with Applications, 38 (7) (2011), pp. 8170-8177, 10.1016/J.ESWA.2010.12.160

View PDFView articleView in ScopusGoogle ScholarAstray et al., 2019