CPS844 Assignment 1

April 7, 2022

Laith Kamal

Abstract

Using various classifications, this study seeks to investigate the Census Income Features Set. There have been many articles published recently about the economic situation in the United States. The economy is often evaluated based on household and individual income (7).

Introduction

When predicting salaries, we're able to use the data set Adult Census Income to see the attributes that can be used to predict the result. We notice that the economic status of an individual is heavily dependent on several factors such as occupation, relationship status, education, and their native country. This is where the Census Database from 1994 comes in, which is a clean records database extracted by Barry Becker (7).

- The data set has **48,842 instances**, of which **32,561** are from the original census data and **16,281** are from the so-called "adult test" file. The test file was obtained by removing some records that had unknown values for some attributes.
- The data set has **15 variables**, of which **14 are attributes** and **1 is a class label**.

There are several classification methods we will be using are Random Forest Classifier, Gradient Boosting, SVM, Naïve Bayes, and K-Neighbors Classifier. Our aim is to find the most accurate classifier that can accurately predict whether a person is making lower or over 50K. Furthermore, to see if we can more accurately predict our result.

Preparing The Data

We first must start by preparing the data and making sure that there are no null values, because Null values can lead to incorrect analyses and predictions, as they represent missing or unknown data and can skew results.

```
In [45]: # Check for missing values
         print(data.isnull().sum())

age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
dtype: int64
```

As we can see there are no null values which means we can continue with working with our data, if needed we could replace null values with mean or mode in order get rid of them.
We know that the target values are strings, with >50K or <50K. For our classification algorithm, we must change these into binary values. The reason for this is that binary classification simplifies the problem and makes it easier for the algorithm to learn. By converting the target variable to binary, the algorithm can better distinguish between the two classes and optimize its predictions accordingly. Through this we'll be able to convert Less than 50K as 1 or greater than or equal to 50K as 0.

```
# Convert the target variable to binary
data['income'] = data['income'].apply(lambda x: x.strip())
data['income'] = data['income'].replace({'>50K': 1, '<=50K': 0})
```

As for the categorical variables we have such as work class, education, and marital status. etc, we will need to convert the categorical variable to numerical. Converting categorical variables to numerical is important for classification models because most machine learning algorithms are designed to work with numerical data. Categorical variables, such as text or categories, cannot be processed by most classification algorithms directly. By converting categorical variables to numerical, the algorithm can better understand and analyze the data, and learn the relationships and patterns between variables.

To do this as simply as possible we will be using Label Encoder, which is a class in the scikit-learn library of Python that is used for converting categorical variables to numerical data. It is a preprocessing technique that assigns a numerical label to each unique category or class in the categorical variable.

Next, we will be splitting the data into training and testing data, it's important to do this because the purpose of splitting the data is to train the model on one subset of data (training set) and then evaluate its performance on another subset of data (testing set) that it has never seen before. This process allows for a more accurate assessment of the model's ability to generalize to new data. If the model is trained and evaluated on the same data, it may perform well on the training data but poorly on new, unseen data. Therefore, splitting the data into training and testing sets helps to ensure the model's accuracy and reliability.

```
# Split the data into training and testing sets
X = data.drop('income', axis=1)
y = data['income']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Classifier 1: Random Forest Classifier

Random Forest Classifier is an ensemble learning method that combines multiple decision trees to create a robust and accurate model for classification tasks. The algorithm starts by randomly selecting a subset of features and data points from the training set, and then builds a decision tree based on the selected features and data points. This process is repeated multiple times to create a forest of decision trees. During the prediction phase, each decision tree in the forest makes a prediction, and the final prediction is determined by taking the majority vote of all the trees in the forest (6).

```
Model: Random Forest
Accuracy: 0.8571867321867321
Confusion Matrix: [[4569  343]
 [ 587 1013]]
Classification Report:               precision    recall  f1-score   support

           0       0.89      0.93      0.91      4912
           1       0.75      0.63      0.69      1600

    accuracy                           0.86      6512
   macro avg       0.82      0.78      0.80      6512
weighted avg       0.85      0.86      0.85      6512
```

The results of the machine learning model using the Random Forest algorithm show an accuracy of 0.857, which is a relatively high accuracy. The confusion matrix shows that the model predicted 4569 true negatives and 1013 true positives, but also 587 false negatives and 343 false positives. The precision score for class 0 is 0.89, meaning that 89% of the predicted negative cases were negative, while the precision score for class 1 is 0.75, meaning that 75% of the predicted positive cases were positive. The recall score for class 0 is 0.93, indicating that the model correctly identified 93% of the actual negative cases, while the recall score for class 1 is 0.63, indicating that the model only identified 63% of the actual positive cases. Overall, the results suggest that the model performed well in predicting negative cases but struggled to accurately predict positive cases.

Classifier 2: Gradient Boosting

Gradient Boosting works by iteratively improving a weak learner or decision tree model. The algorithm starts by creating a single decision tree, which may not perform well. The algorithm then focuses on the errors or residuals of the first model and builds a second tree to predict these errors. This second model is added to the first model to make a better prediction. The algorithm then continues to build more trees,

each one focused on the errors of the previous model until the desired number of trees or a stopping criterion is reached. During each iteration, the algorithm uses a gradient descent optimization technique to minimize the residual error between the predictions and the actual target values. The final prediction is a combination of all the decision trees, where each tree's prediction is weighted according to its accuracy (4).

```
Model: Gradient Boosting
Accuracy: 0.8657862407862408
Confusion Matrix: [[4650  262]
 [ 612  988]]
Classification Report:               precision    recall  f1-score   suppor

           0       0.88      0.95      0.91      4912
           1       0.79      0.62      0.69      1600

    accuracy                           0.87      6512
   macro avg       0.84      0.78      0.80      6512
weighted avg       0.86      0.87      0.86      6512
```

The Gradient Boosting model achieved an accuracy of 0.865, which indicates that it correctly predicted 86.5% of the instances in the test data set. The Confusion Matrix shows that the model correctly classified 4650 instances as negative and 988 instances as positive, while misclassifying 262 instances as false negatives and 612 instances as false positives. The Classification Report indicates that the model performed better in predicting negative instances (precision: 0.88, recall: 0.95, f1-score: 0.91) than positive instances (precision: 0.79, recall: 0.62, f1-score: 0.69). The macro-average F1-score of 0.80 suggests that the model is reasonably good at both positive and negative predictions. Overall, the model's performance seems good, but further analysis is needed to determine whether it is suitable for the intended application.

Classifier 3: SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression. It works by finding the optimal hyperplane that maximizes the margin between classes. It can handle non-linearly separable data by transforming it into a higher dimensional space using a kernel function. SVM can also handle multi-class classification problems using a one-vs-one or one-vs-all approach. SVM is a powerful technique for classification, but it can be computationally expensive for large datasets, and the choice of kernel function and parameters can affect its performance (3).

```
Model: SVM
Accuracy: 0.8433660933660934
Confusion Matrix: [[4609  303]
 [ 717  883]]
Classification Report:               precision    recall  f1-score   support

           0       0.87      0.94      0.90      4912
           1       0.74      0.55      0.63      1600

    accuracy                           0.84      6512
   macro avg       0.80      0.75      0.77      6512
weighted avg       0.84      0.84      0.83      6512
```

The SVM model achieved an accuracy of 0.843, correctly predicting 84.3% of the instances in the test data set. The Confusion Matrix shows that the model correctly classified 4609 instances as negative and 883 instances as positive, while misclassifying 303 instances as false negatives and 717 instances as false positives. The Classification Report indicates that the model performed better in predicting negative instances (precision: 0.87, recall: 0.94, f1-score: 0.90) than positive instances (precision: 0.74, recall: 0.55, f1-score: 0.63). The macro-average F1-score of 0.77 suggests that the model's performance is satisfactory but not as good as the Gradient Boosting model.

Classifier 4: Naïve Bayes

Naive Bayes is a supervised machine learning algorithm used for classification. It's based on Bayes' theorem, which describes the probability of an event based on prior knowledge. Naive Bayes assumes that the features are independent of each other, which is often not true, but still, it's a fast and effective algorithm (2).

```
Model: Naive Bayes
Accuracy: 0.8066646191646192
Confusion Matrix: [[4691  221]
 [1038  562]]
Classification Report:               precision    recall  f1-score   support

           0       0.82      0.96      0.88      4912
           1       0.72      0.35      0.47      1600

    accuracy                           0.81      6512
   macro avg       0.77      0.65      0.68      6512
weighted avg       0.79      0.81      0.78      6512
```

The Naive Bayes model achieved an accuracy of 0.807, correctly predicting 80.7% of the instances in the test data set. The Confusion Matrix shows that the model correctly classified 4691 instances as negative and 562 instances as positive, while misclassifying 221 instances as false negatives and 1038

instances as false positives. The Classification Report indicates that the model performed better in predicting negative instances (precision: 0.82, recall: 0.96, f1-score: 0.88) than positive instances (precision: 0.72, recall: 0.35, f1-score: 0.47). The macro-average F1-score of 0.68 suggests that the model's performance is not as good as the Gradient Boosting or SVM models. Naive Bayes is a simple and efficient algorithm but is known for its overly simplistic assumption of feature independence, which can affect its performance in practice.

Classifier 5: K- Neighbors Classifiers

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for classification and regression tasks. It works by assigning an instance to the class most common among its k nearest neighbors in a training data set. The value of k is a hyperparameter that determines the number of neighbors to consider (5).

```
Model: K-Neighbors Classifier
X:\Softwares\Conda\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sc
iPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the s
tatistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False
to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
Accuracy: 0.8194103194103194
Confusion Matrix: [[4441  471]
 [ 705  895]]
Classification Report:               precision    recall  f1-score   support

           0       0.86      0.90      0.88      4912
           1       0.66      0.56      0.60      1600

    accuracy                           0.82      6512
   macro avg       0.76      0.73      0.74      6512
weighted avg       0.81      0.82      0.81      6512
```

The K-Neighbors Classifier algorithm achieved an accuracy of 0.819, which is slightly lower than some of the other algorithms tested. The confusion matrix shows that there were 471 false positives and 705 false negatives, indicating that the model struggled with correctly identifying positive cases. The precision for positive cases was 0.66, which means that when the model predicted a positive case, it was correct 66% of the time. The recall for positive cases was 0.56, which means that the model correctly identified 56% of all positive cases in the dataset. Overall, while the accuracy is respectable, the model's performance in correctly identifying positive cases is not as strong as some of the other algorithms tested.

Algorithm	Accuracy
Random Forest	0.857
Gradient Boosting	0.866

Support Vector Machine	0.843
Naive Bayes	0.807
K-Neighbors Classifier	0.819

Based on the results, Gradient Boosting has the highest accuracy score of 0.866, closely followed by Random Forest with an accuracy of 0.857. SVM and K-Neighbors Classifier had similar scores of 0.843 and 0.819, respectively. Naive Bayes had the lowest accuracy score of 0.807. Overall, it can be concluded that Gradient Boosting and Random Forest performed the best on the given dataset, while Naive Bayes performed the worst.

This suggests that:

- Gradient boosting is the most suitable algorithm for this dataset, as it can handle both categorical and numerical features well and reduce overfitting by combining multiple weak learners.
- Random forest is also a good choice as it can capture complex interactions among features and handle imbalanced data well.
- Support vector machines can perform well on linearly separable data but may struggle with high-dimensional or noisy data.
- Naive bayes is a simple and fast algorithm but it makes strong assumptions about the independence of features which may not hold.
- K-neighbors classifier is a lazy learning algorithm that does not require training, but it can be affected by outliers and irrelevant features.

Attribute Selection Algorithms

The algorithms used are SelectKBest with f_classif, SelectKBest with mutual_info_classif, and Recursive Feature Elimination (RFE) with Logistic Regression estimator. The k value is set to 5 for this task.

Feature selection:

Feature selection is the process of selecting the most relevant features or attributes that contribute the most towards the prediction of the target variable in a machine learning model. The main goal of feature selection is to reduce the dimensionality of the data while maintaining or improving the accuracy of the predictive model (1) .

Importance of Feature Selection:

Feature selection is an important step in machine learning because it helps in reducing the computational cost and the risk of overfitting the model. Overfitting occurs when the model becomes too complex and fits the training data very well, but fails to generalize to new, unseen data. By reducing the number of features, feature selection reduces the complexity of the model and helps in achieving a good trade-off between bias and variance.

Techniques for Feature Selection

There are various techniques for feature selection, such as filter methods, wrapper methods, and embedded methods.

1. Filter methods involve selecting features based on some statistical measure of their correlation with the target variable. For example, correlation coefficient, mutual information, or chi-squared test can be used as the statistical measure. The selected features are then used to train the model.

2. Wrapper methods involve selecting features based on the performance of the model on a subset of features. It involves iterating over all possible feature subsets and selecting the subset that results in the best model performance. The selected subset is then used to train the model.
3. Embedded methods involve selecting features during the model training process. The feature selection is integrated into the model training algorithm, which selects the most relevant features based on some criterion, such as regularization or decision tree splitting criterion.

SelectKBest with f_classif:

SelectKBest with f_classif is a feature selection method used in machine learning that selects the top k features based on their scores in the ANOVA F-test. It is a filter method that evaluates the linear relationship between each feature and the target variable. The F-test is used to determine whether the means of the groups are significantly different from each other. The higher the F-value, the more significant the relationship between the feature and the target variable.

The SelectKBest function from the scikit-learn library is used to implement this method. It takes two arguments - the scoring function and the number of features to select (k). In the case of SelectKBest with f_classif, the scoring function is f_classif, which stands for F-value between label/feature for classification tasks.

Select KBest with mutual_info_classif:

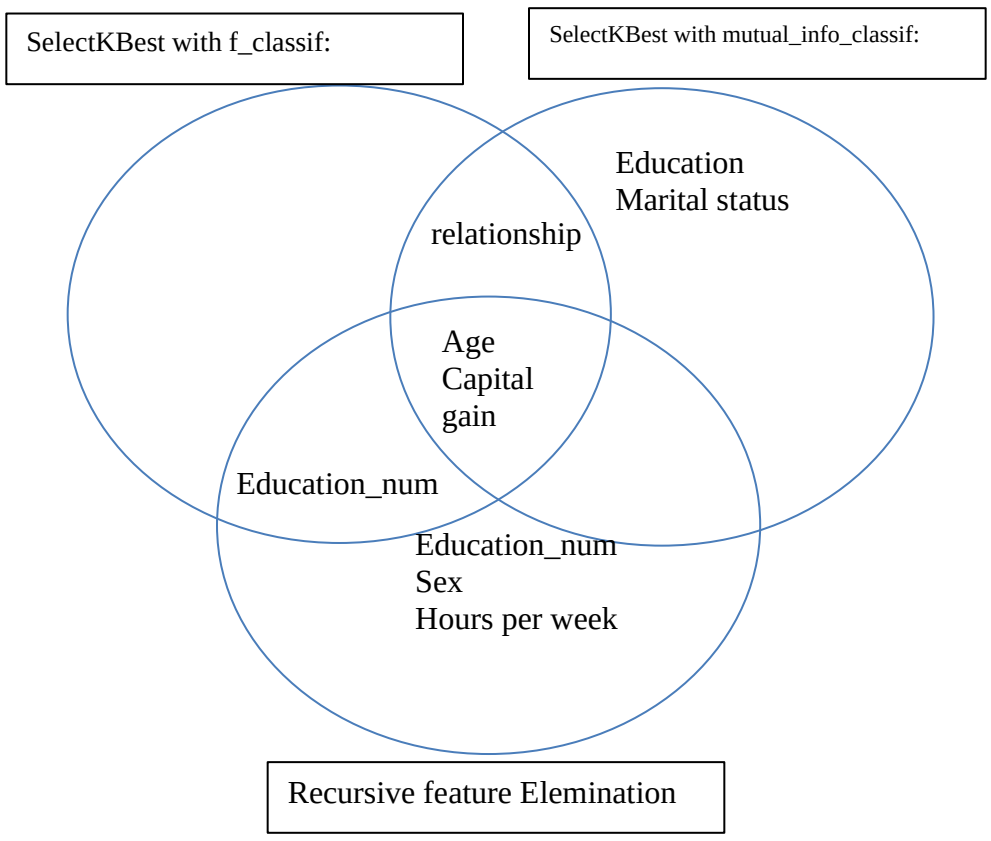
SelectKBest with mutual_info_classif is a feature selection algorithm used in machine learning to select the k best features from a given dataset based on their mutual information score with the target variable. Mutual information is a measure of the dependence between two variables, and in this case, it is used to evaluate the non-linear relationship between each feature and the target variable. The SelectKBest with mutual_info_classif algorithm works by ranking the features based on their mutual information score with the target variable and selecting the k best features with the highest scores.

Recursive Feature Elimination (RFE):

Recursive Feature Elimination (RFE) is a feature selection algorithm used in machine learning to select the most important features from a given dataset by recursively removing features based on their importance. RFE is a wrapper method that involves training a model on the full set of features, ranking the features based on their importance, and recursively eliminating the least important features until the desired number of features is reached.

RFE works by first training a model on the entire set of features, and then ranking the features based on their importance in the model. The least important feature is then eliminated, and the model is retrained on the remaining features. This process is repeated until the desired number of features is reached or until a stopping criterion is met.

As we can see, here are the most important features selected by each feature selection algorithm:



```
Attribute Selector: SelectKBest-f_classif
Selected Features: Index(['age', 'education-num', 'relationship', 'capital-gain',
       'hours-per-week'],
      dtype='object')
--------------------------------------------------
Attribute Selector: SelectKBest-mutual_info_classif
Selected Features: Index(['age', 'education', 'marital-status', 'relationship', 'capital-gain'], dtype='object')
--------------------------------------------------
Attribute Selector: Recursive Feature Elimination
Selected Features: Index(['age', 'education-num', 'sex', 'capital-gain', 'hours-per-week'], dtype='object')
--------------------------------------------------
```

Based on the outputs, we can see that the 'age', and 'capital-gain' features were selected by all three selectors, while other features varied. Therefore, we can conclude that these two features are the most important features for predicting the target variable in this dataset. From the output, we can see that the accuracy of the models with selected attributes is generally lower than the accuracy of the models with all attributes. However, the difference in accuracy varies between the attribute selection algorithms and the models.

The **SelectKBest-f_classif algorithm** selects 'age', 'education-num', 'relationship', 'capital-gain', and 'hours-per-week' as the most important features. When we use these features for model training, we can see that the accuracy of the Random Forest model drops from 0.857 to 0.835, the accuracy of the Gradient Boosting model drops from 0.866 to 0.854, and the accuracy of the SVM model drops from 0.843 to 0.845. However, the Naive Bayes model and the K-Neighbors Classifier model also see a drop in accuracy from 0.807 to 0.792 and from 0.819 to 0.829, respectively.

On the other hand**, the SelectKBest-mutual_info_classif algorithm** selects 'age', 'education', 'marital-status', 'relationship', and 'capital-gain' as the most important features. When we use these features for model training, we can see that the accuracy of the Random Forest model drops from 0.858 to 0.838, the accuracy of the Gradient Boosting model drops from 0.866 to 0.852, the accuracy of the SVM model

drops from 0.843 to 0.811, and the accuracy of the K-Neighbors Classifier model drops from 0.819 to 0.818. However, the Naive Bayes model sees a slight increase in accuracy from 0.807 to 0.791.

Finally, the **Recursive Feature Elimination** algorithm selects 'age', 'education-num', 'sex', 'capital-gain', and 'hours-per-week' as the most important features. When we use these features for model training, we can see that the accuracy of the Random Forest model drops from 0.857 to 0.813, the accuracy of the Gradient Boosting model drops from 0.866 to 0.835, the accuracy of the SVM model drops from 0.843 to 0.827, and the accuracy of the K-Neighbors Classifier model drops from 0.819 to 0.801. However, the Naive Bayes model sees a slight increase in accuracy from 0.807 to 0.793.

```
Results with all attributes:
Model: Random Forest
Accuracy: 0.8573402948402948
Model: Gradient Boosting
Accuracy: 0.8657862407862408
Model: SVM
Accuracy: 0.8433660933660934
Model: Naive Bayes
Accuracy: 0.8066646191646192
Model: K-Neighbors Classifier
X:\Softwares\Conda\lib\site-package
cally preserves the axis it acts al
liminated, and the value None will
  mode, _ = stats.mode(_y[neigh_ind
Accuracy: 0.8194103194103194
```

Results with all attributes

```
Results with selected attributes:
Attribute Selector: SelectKBest-f_classif
Model: Random Forest
Accuracy: 0.8347665847665847
Model: Gradient Boosting
Accuracy: 0.8538083538083538
Model: SVM
Accuracy: 0.8450552825552825
Model: Naive Bayes
Accuracy: 0.7922297297297297
Model: K-Neighbors Classifier
X:\Softwares\Conda\lib\site-packages\sklearn\
cally preserves the axis it acts along. In Sc
liminated, and the value None will no longer
  mode, _ = stats.mode(_y[neigh_ind, k], axis
Accuracy: 0.8286240786240786
```

Results with SelectKBest-f_classif

```
Attribute Selector: SelectKBest-mutual_info_classif
Model: Random Forest
Accuracy: 0.8378378378378378
Model: Gradient Boosting
Accuracy: 0.8524262899262899
Model: SVM
Accuracy: 0.8106572481572482
Model: Naive Bayes
Accuracy: 0.7911547911547911
Model: K-Neighbors Classifier
X:\Softwares\Conda\lib\site-packages\sklearn\neighbors\
cally preserves the axis it acts along. In SciPy 1.11.0
liminated, and the value None will no longer be accepte
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
Accuracy: 0.8180282555282555
```

Results with SelectKBest-mutual_info_classif

```
Attribute Selector: Recursive Feature Elimination
Model: Random Forest
Accuracy: 0.812960687960688
Model: Gradient Boosting
Accuracy: 0.8353808353808354
Model: SVM
Accuracy: 0.8267813267813268
Model: Naive Bayes
Accuracy: 0.792997542997543
Model: K-Neighbors Classifier
X:\Softwares\Conda\lib\site-packages\sklearn\neighbor
cally preserves the axis it acts along. In SciPy 1.11
liminated, and the value None will no longer be accep
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
Accuracy: 0.8014434889434889
```

Results with Recursive Feature Elimination

In conclusion, from the results, we can see that attribute selection can help to improve the accuracy of some models and degrade the accuracy of some other models. Attribute selection may decrease the accuracy of a model in some cases because it may remove important features that contribute to the prediction. Feature selection algorithms typically select features based on some statistical measure such as correlation, mutual information, or significance test, and they assume that these measures accurately reflect the importance of the features. However, these assumptions may not always hold true causing a decrease in accuracy.

Conclusion and Further possibilities

In conclusion, we can see that the best result we could have been using gradient boosting which is 86.57% accuracy for our model. This is a respectable result as we know that there are many more metrics that could be used to evaluate a person's income. I believe that the data set could be a bit more descript able in terms of the occupation. For example, if we can specify the industry that the individual works in, that would make our prediction perhaps much stronger since for example individuals in the tech industry could be making much more than individuals in the furniture business or sales.

We also noticed that feature selection often does not improve the accuracy of the model, but could also decrease it, this can be attributed to several things such as information loss, where some features are removed that might seem irrelevant on their own but are crucial when paired up with others. This could be the case because we do not have many features in this data set. Which might have simplified the model and perhaps lead to overfitting, where the model becomes too specific to the training data and does not generalize well to new data.

Works Cited:

Data: https://archive.ics.uci.edu/ml/datasets/Census+Income

1) Brownlee, Jason. "An Introduction to Feature Selection." *MachineLearningMastery.com*, 28 June 2021, https://machinelearningmastery.com/an-introduction-to-feature-selection/.
2) Gandhi, Rohith. "Naive Bayes Classifier." *Medium*, Towards Data Science, 17 May 2018, https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c.
3) Gandhi, Rohith. "Support Vector Machine - Introduction to Machine Learning Algorithms." *Medium*, Towards Data Science, 5 July 2018, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.
4) Saini, Anshul. "Gradient Boosting Algorithm: A Complete Guide for Beginners." *Analytics Vidhya*, 14 Oct. 2021, https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/.
5) Shafi, Adam. "K-Nearest Neighbors (KNN) Classification with Scikit-Learn." *DataCamp*, DataCamp, 2 Aug. 2018, https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn.
6) "Sklearn.ensemble.randomforestclassifier." *Scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
7) *UCI Machine Learning Repository: Census Income Data Set*, https://archive.ics.uci.edu/ml/datasets/Census+Income.