Machine Learning Engineer Nano degree

Capstone Project

Bike Buyer Prediction

Sanke Sneha

February 21, 2019

Definition

Project overview:

Bikes are the affordable vehicles for all kinds of people. As the need of transportation increases the usage of bikes also increased which results in the sudden rise of automobile industries. The automotive industry continues to face set of challenges. Most manufacturing operations in automotive industries are still largely dependent on experiences-based human decisions. I am highly interested to apply Machine Learning in the automotive industry to make a remarkable ability to bring out hidden relationships among datasets and make predictions.

The dataset that I have used is my private data and can be found on Bike Buyer Dataset

So my goal is to predict the bike buyers.

The related academic work is found at Bike sharing project . It is somewhat similar to my project and I got inspired by this to apply machine learning in automotive industry as well. In this article it is taken as regression task and my task follows classification.

Problem Statement:

My main aim is to predict the bike buyers. For doing this I selected my private dataset and that can be found in the following link Bike Buyer Dataset

So the goal is to predict the bike buyers. Here I am using classification models for this problem. The techniques I am going to use in this is svm classifier, decision tree classifier, random forest classifier and find the f-score of each model and select the best model with high f-score to predict the bike buyer or not. Here the input is training data that we took and output will be whether bike buyer or not.

The tasks involved in it are:

1. Download the data
2. Cleaning the data and removing the null data points, duplicated data rows.
3. Visualizing the data. For visualization of data I have used histogram, pair plot and heatmap.
4. Split the data and train and test which classifier performs good on the dataset based on

the evaluation metric we have chosen.

Metrics:

I want to use accuracy, precision and f-score as evaluation metric for prediction of bike buyer. Here I am predicting f-score for selected models because the dataset is unbalanced with [5997] are 'non-buyers' and [998] are 'buyers'. Here we will select a model whose f-score is greater than all the other models and we treat it as the best.

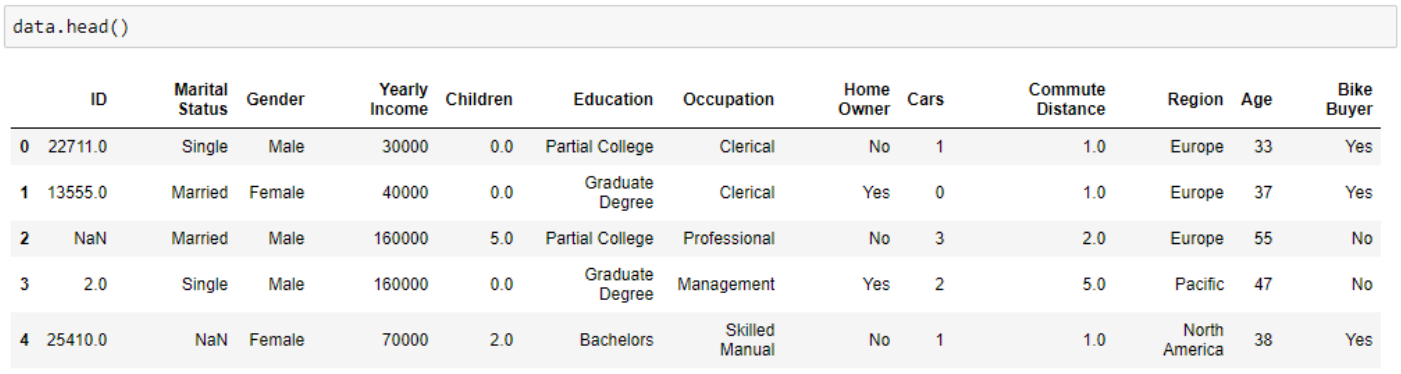
F_1 score (also **F-score** or **F-measure**) is a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score.

$$F_1 = \left(\frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2}\right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Analysis

Data Exploration:

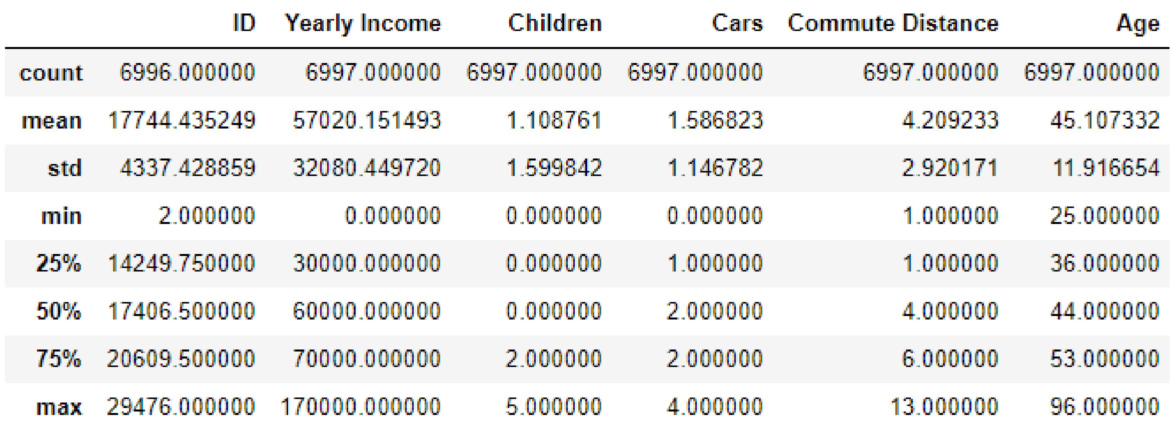
The dataset is quite interesting because it is a good mixture of categorical and numerical attributes. There are total 13 attributes and 7000 instances are there. There are few missing values also. There are mainly two classes' bike buyers or not. The dataset I am using is unbalanced because among 7000 customers [5997] are 'non-buyers' and [998] are 'buyers'.



```
data.head()
```

	ID	Marital Status	Gender	Yearly Income	Children	Education	Occupation	Home Owner	Cars	Commute Distance	Region	Age	Bike Buyer
0	22711.0	Single	Male	30000	0.0	Partial College	Clerical	No	1	1.0	Europe	33	Yes
1	13555.0	Married	Female	40000	0.0	Graduate Degree	Clerical	Yes	0	1.0	Europe	37	Yes
2	NaN	Married	Male	160000	5.0	Partial College	Professional	No	3	2.0	Europe	55	No
3	2.0	Single	Male	160000	0.0	Graduate Degree	Management	Yes	2	5.0	Pacific	47	No
4	25410.0	NaN	Female	70000	2.0	Bachelors	Skilled Manual	No	1	1.0	North America	38	Yes

For the best result we will split the data into training set and testing set. On a whole we will assign 80% of the data to training set and 20% of the data to testing set. The information about the numerical attributes is as follows



	ID	Yearly Income	Children	Cars	Commute Distance	Age
count	6996.000000	6997.000000	6997.000000	6997.000000	6997.000000	6997.000000
mean	17744.435249	57020.151493	1.108761	1.586823	4.209233	45.107332
std	4337.428859	32080.449720	1.599842	1.146782	2.920171	11.916654
min	2.000000	0.000000	0.000000	0.000000	1.000000	25.000000
25%	14249.750000	30000.000000	0.000000	1.000000	1.000000	36.000000
50%	17406.500000	60000.000000	0.000000	2.000000	4.000000	44.000000
75%	20609.500000	70000.000000	2.000000	2.000000	6.000000	53.000000
max	29476.000000	170000.000000	5.000000	4.000000	13.000000	96.000000

The interpretations that I have made about the data is

from the given data the bike buyers who are willing to buy are less in number
More females(521) than males(477)
Married(504) customers are more willing to buy than singles(494).
Females who are single(272) are more willing to buy.
The customers in North America are more willing to buy when compared to other regions.
Males who are married in North America(131) are more willing to buy.

Attributes:

ID

Marital Status: Married or Single

Gender: Female or male

Yearly Income: range of income that a customer is having

Children: number of children of the customer

Education: Qualification of the customer

Occupation: occupation of the customer

Home Owner: having a own house or not

Cars: Number of cars

Commute Distance: distance from home to work place

Region: Place of the customer

Age: age of the customer

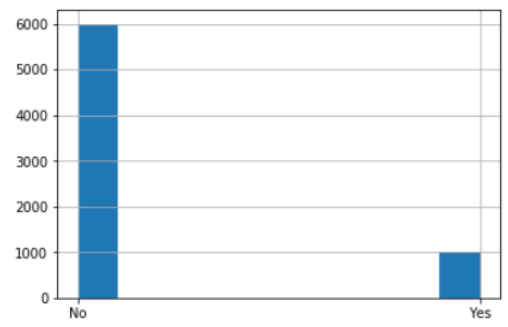
Bike Buyer: yes or no

Exploratory Visualization:

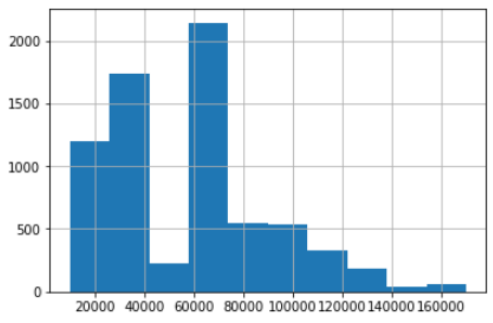
Histogram is a graphical representation of the Distribution of data

- Bins: the intervals used in a histogram. The data must be separated into mutually exclusive and exhaustive bins

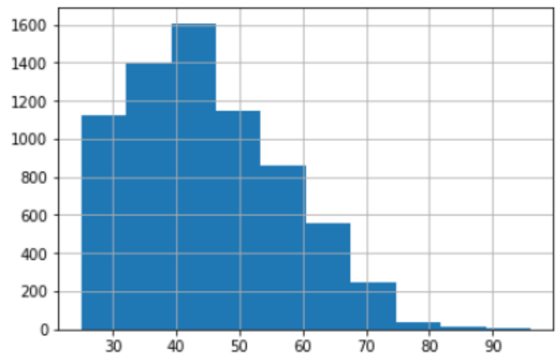
It is mainly used to represent the relationship between data. In my project I used histogram to know the different characteristic of each attribute and also relation among them.
Some of the visuals that I have made using histogram are as follows



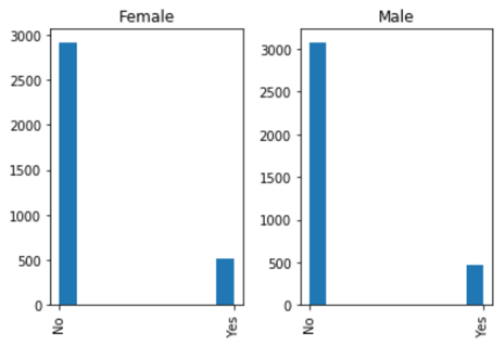
Bike Buyer



Yearly Income

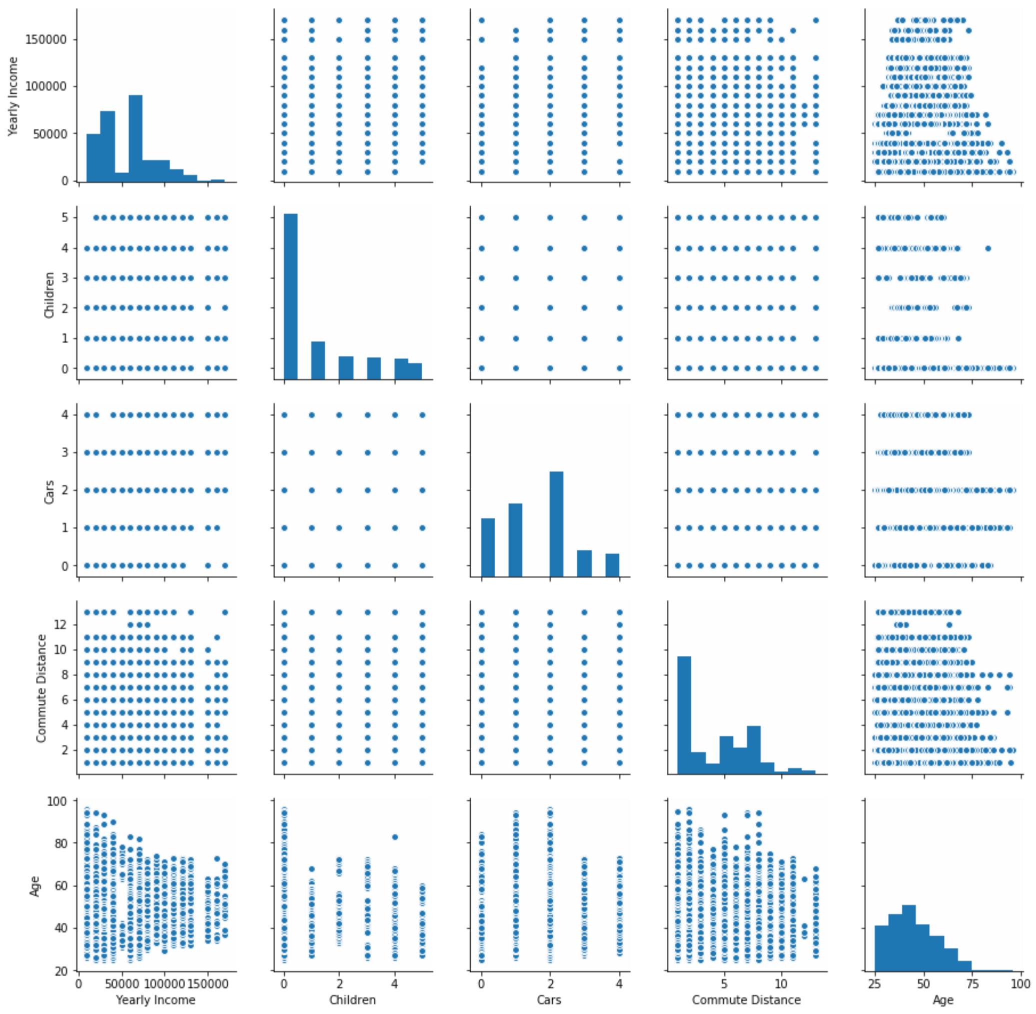


Age



Bike Buyer

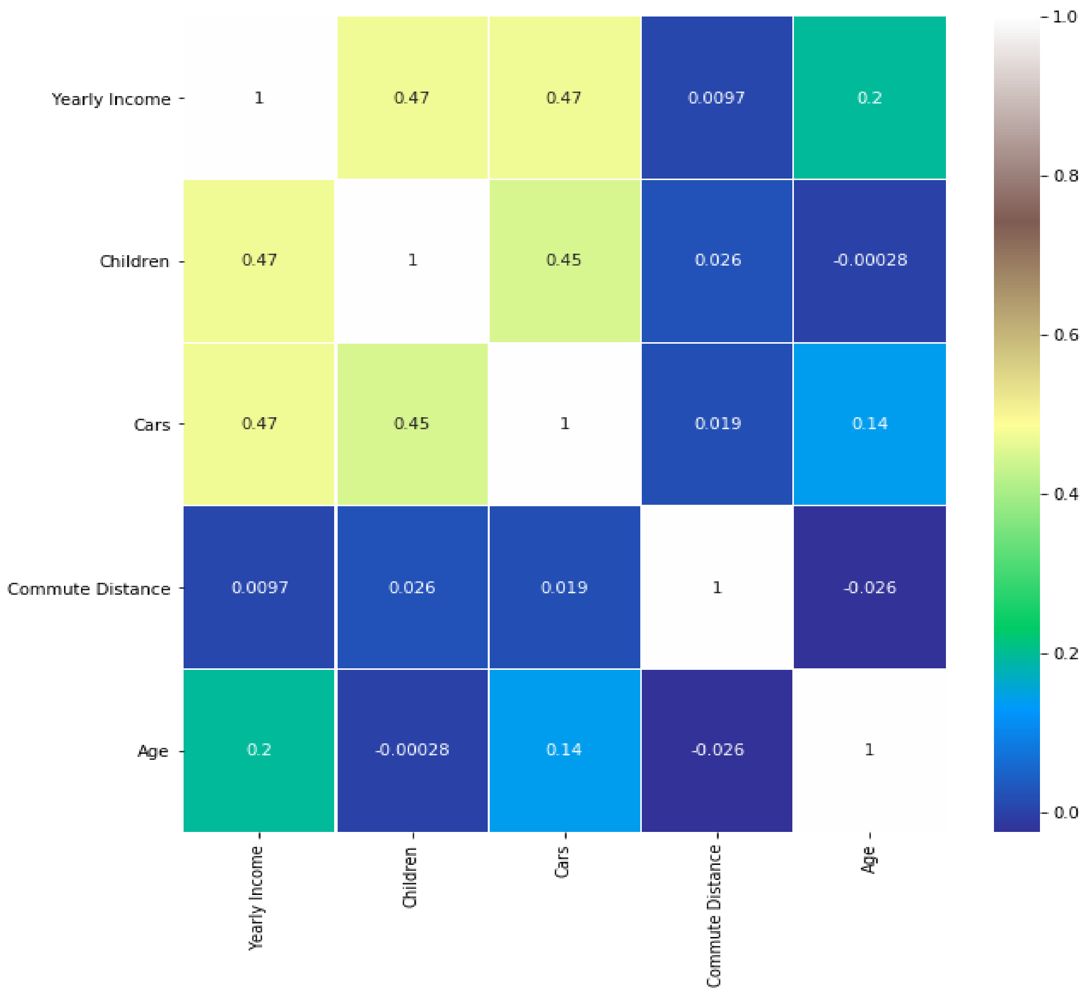
First graph shows that there are more customers who are no bike buyers in the dataset.
Second graph shows the yearly income range of all the customers in the dataset.
Third graph shows that there are mostly 40 to 50 aged people in the dataset.
Fourth graph shows that more females are willing to buy bikes when compared to males.



From the above pair plot we cannot clearly visualize the correlation between the numerical attributes so we will go with the heatmap to know clearly about the correlation.

Heatmap:

The heatmap is a 2-D representation of data in which values are represented by colors. A simple heatmap provides the immediate visual summary of information. More elaborate heatmaps allow the user to understand complex data.



From the above heatmap we will represent the correlation of the numerical attributes in the data. The above heatmap clearly visualize the correlation between the numerical attributes. We can say that there is no much correlation between them.

Algorithms and techniques:

Decision Trees: Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predict the value.

Some advantages of decision trees are:

Simple to understand and to interpret. Trees can be visualized. Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information. Able to handle multi-output problems. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over fitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement. There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Parameters:

Class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False

Random Forest:

Tree models are known to be high variance, low bias models. In consequence, they are prone to overfit the training data. This is catchy if we recapitulate what a tree model does if we do not prune it or introduce early stopping criteria like a minimum number of instances per leaf node. Well, it tries to split the data along the features until the instances are pure regarding the value of the target feature, there are no data left, or there are no features left to spit the dataset on. If one of the above holds true, we grow a leaf node. The consequence is that the tree model is grown to the maximal depth and therewith tries to reshape the training data as precise as possible which can easily lead to over fitting. Another drawback of classical tree models like the (ID3 or CART) is that they are relatively unstable. This instability can lead to the situation that a small change in the composition of the dataset leads to a completely different tree model.

Parameters:

Class sklearn.ensemble.RandomForestClassifier (n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None

Advantages:

1. Reduction in over fitting: by averaging several trees, there is a significantly lower risk of over fitting.
2. Less variance: By using multiple trees, you reduce the chance of stumbling across a classifier that doesn't perform well because of the relationship between the train and test data.

Disadvantages :

1. It takes more time to train samples.

KNN:

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output

2. Calculation time
3. Predictive Power

Advantages:
The cost of the learning process is zero No assumptions about the characteristics of the concepts to learn have to be done Complex concepts can be learned by local approximation using simple procedures

Disadvantages:
The model can not be interpreted (there is no description of the learned concepts) It is computationally expensive to find the k nearest neighbours when the dataset is very large Performance depends on the number of dimensions that we have (curse of dimensionality) ==⇒ Attribute Selection

Parameters:
class sklearn.neighbors.KNeighborsClassifier(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*, ***kwargs*)

Support vector machine (svm):

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.
To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Advantages:
SVM's are very good when we have no idea on the data. Works well with even unstructured and semi structured data like text, Images and trees. The kernel trick is real strength of SVM. With an appropriate kernel function, we can solve any complex problem. Unlike in neural networks, SVM is not solved for local optima. It scales relatively well to high dimensional data. SVM models have generalization in practice; the risk of overfitting is less in SVM.

Disadvantages:
Choosing a good kernel function is not easy. Long training time for large datasets. Difficult to understand and interpret the final model, variable weights and individual impact. Since the final model is not so easy to see, we can not do small calibrations to the model hence it's tough to incorporate our business logic.

Parameters:
class sklearn.svm.**SVC**(*C=1.0*, kernel='rbf', degree=*3*, gamma='auto_deprecated', *coef0=0.0*, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=Fals*e*, max_iter=-1, decision_function_shape=*'ovr'*, random_state=*None*

Benchmark Model:

Benchmark model is a model which we will take as reference and achieve the best than the benchmark model. In our project, we will take svm classifier as benchmark model. Using knn we will achieve fscore 0.2. Now we will try and achieve the better f1_score than the benchmark model.

Methodology

Data Pre-processing:

In this step of data pre-processing we will pre-process the data. We will read the data from dataset and replace the null values.

We will know the information about all the data types. We will know the mean standard deviation and various metrics regarding to the numerical data. To know the correlation between the numerical attributes we will plot the graphs to visualize the data (this context we will use pair plot and heatmap).

we need to drop the ID column which is unnecessary. And we also need to drop the rows where income value is 0. And inorder to remove the missing values in each column I have replaced them with most frequent and mean, median values.

We will now remove the true outliers. We will consider the data which is three standard deviations away from the mean as the outliers and remove them.

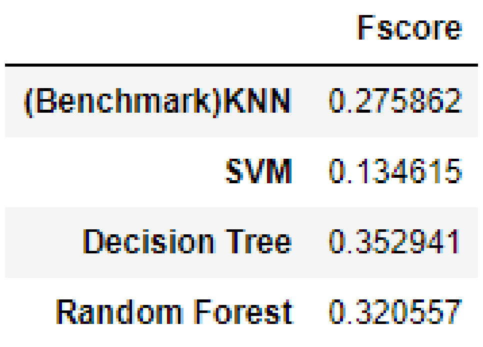
Now we need to perform a One Hot Encoding of the categorical variables to prepare the data for classification. We can do this easily by using OneHotEncoder from the sklearn.preprocessing module. Normalising to rescale the features to a standard range of values using MinMaxScaler

After that the whole dataset is divided into training and testing data using train_test_split from sklearn.model_selection.

Implementation:

Here we will initialize each classifier and fit the model with training data and then predict the output by using predict() function. The version of the sklearn learn library I used is 0.20. So the hyperparameters for each algorithm were left at the default settings.

We will find out the accuracy, precision, recall, f1_score for the all models. We will now choose the best model out of svm, random forest, decision tree whose f1_score will be more than that of the benchmark model.



	Fscore
(Benchmark)KNN	0.275862
SVM	0.134615
Decision Tree	0.352941
Random Forest	0.320557

Among the above reports Decision Tree seems to be performing well.

While building the Decision Tree Classifier I faced a complication while tuning the parameters. Firstly we should come to a conclusion about the parameters we are going to tune. Then by varying the range of values of the chosen parameter we should be able to get the parameter value which gives us the better f1_score than the untuned value. So it is complicated to choose the parameter that is to be tuned in this case max_depth and random_state.

Refinement

I found out Decision Tree as the best classifier out of the chosen classifiers. Now we will perform tuning of Decision Tree classifier in order to achieve the better f1_score. Here we will use GridSearchCV(). Here we will give max_depth= range(1,20) and random_state=100 as parameters for GridSearchCV().

The f1_score of tuned Decision tree is 0.3641, which is better than the untuned Decision Tree.

To make the model better I used GridSearchCV() with max_depth and random_state as parameters and I have got the values which gives better performance after varying multiple times with different range of values.

Why I used grid search technique particularly means **it** generates a set of models (which differ from each other in their parameter values, which lie on a **grid**). **Grid**-**search** is a way to select the best of a family of models, parametrized by a **grid** of parameters.

Result

Model evaluation and validation

The final model we have chosen is tuned decision tree which gave us more f1_score that is 0.3641. In order to achieve this f1_score we assigned max_depth=range(1, 20) and random_state=100 but without using the max_depth we achieved the f1_score of only 0.3529 which is a bit less than the tuned value. Here we can say that the solution is reasonable because we are getting less accuracy while using other models.

The final model that is tuned decision tree. This model is also robust enough for the given problem. I tested the Decision Tree for various random states and I can clearly see that there is no big change in the f1_score. I have tested for the random states 100, 200, 300 and I have achieved better f1_score of random state =100. So from that we can say that Small changes in the training data will not affect the results greatly. So the results found from this model can be trusted.

Justification

My final model's solution is better than the benchmark model.

	Tuned Decision Tree	Benchmark model
F1_score	0.2758	0.3641

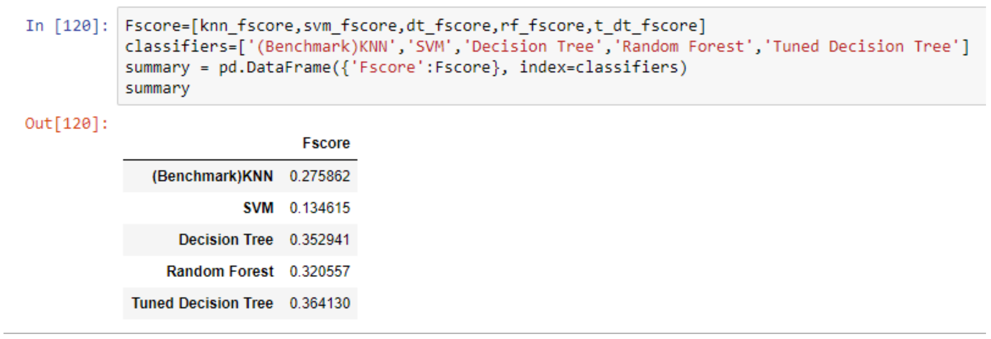
From the above we can conclude that the results for the final model are stronger than the benchmark model.

Hence we can say that tuned decision tree provides the significant solution for the problem of prediction of bike buyer or not.

Conclusion

Free-form Visualization

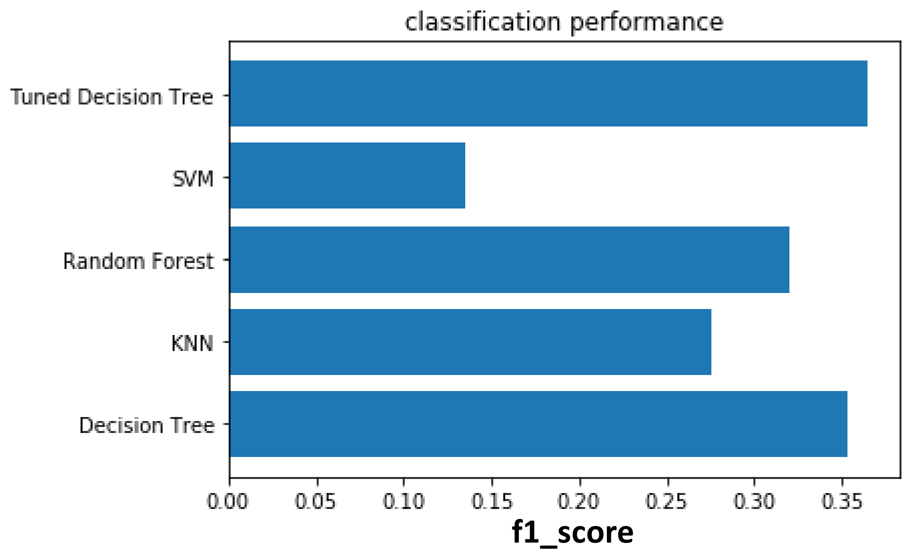
After studying the data and applying all the classification models we can visualize the following



```
In [120]: Fscore=[knn_fscore,svm_fscore,dt_fscore,rf_fscore,t_dt_fscore]
          classifiers=['(Benchmark)KNN','SVM','Decision Tree','Random Forest','Tuned Decision Tree']
          summary = pd.DataFrame({'Fscore':Fscore}, index=classifiers)
          summary
```

Out[120]:

	Fscore
(Benchmark)KNN	0.275862
SVM	0.134615
Decision Tree	0.352941
Random Forest	0.320557
Tuned Decision Tree	0.364130



The interesting part of this project for me is visualizing and making correct interpretations about the data.
The challenge that I faced throughout the process is to select correct values for tuning parameters and I overcame it after doing many trials with different range of values.

Improvements:

Engineer and add more relevant features to the original dataset since Decision Tree has a strong learning capability and a rich dataset might improve its prediction performance.

Reflection:

In this project first I have preprocessed the data to make data clean to use after that I have splitted the data into 80% training and 20% testing data for the best results. After that I have used SVM, Decision tree and random forest algorithm to build the model. After the models got trained by the training data I have calculated the f1_score of each model and compared with the benchmark mark model which I have choosen is knn with 0.27 f1_score. And the model which f1_score is high is selected as best model for the problem and also done with refinement to the model using gridsearchcv() inorder to increase the score of the model.