Classification: Loan Application

Introduction / Description of Dataset

The dataset I am using contains information about loan applicants. My idea is that I should be able to predict if a loan application got accepted or not by the attributes. The class label I am predicting is Loan Status, which is either 'Yes' or 'No'.

The dataset contains nominal/binary attributes 'Married', 'Gender', 'Education', 'Self Employed' and 'Property Area'. It also contains discrete attributes 'Loan ID', 'Dependents', 'Applicant Income', 'Co Applicant Income' and 'Loan Amount'.

I plan on using a Decision Tree and K-Nearest-Neighbour algorithms, and comparing the results to see which is more accurate in this case.

I predict that attributes 'Gender', 'Self Employed' and 'Property Area' will have little or no effect on whether a loan is accepted or not. I also think that attributes 'Credit History' and 'Applicant Income' will have the biggest impact on whether loans are accepted.

Analysis - Nominal/Binary Attributes

	Gender	Married	Dependents	Education	Self_Employed	Loan Amount Term	Credit_History	Property_Area
Count	601	611	599	614	582	600	564	614
Unique	2	2	4	2	2	10	2	3
Mode	Male	Yes	0	Graduate	No	360	1	Semiurban

Freq	489	398	345	480	500	512	475	233

The mode is the only means of finding the ‘centre’ of the nominal attributes, given I cannot use the mean and median for non-numeric data. The frequency row shows the number of times the mode of each feature column appears.

8.14% of the attribute Credit_History’s values are missing (50 out of 614). However, as shown below, it is an important attribute in deciding whether a loan gets accepted or not, so I decided to fill in the missing values with the mode of the attribute.

I grouped each attribute, and found the percentage of time that the attribute’s values result in the loan being accepted.

Gender	Loan_Status
Male	0.693252
Female	0.669643

Married	Loan_Status
Yes	0.716080
No	0.629108

Dependents	Loan_Status
2	0.752475
0	0.689855
1	0.647059
3+	0.647059

Education	Loan_Status
Graduate	0.708333
Not Graduate	0.611940

Self_Employed	Loan_Status
No	0.686000
Yes	0.682927

Credit_History	Loan_Status
1.0	0.795789
0.0	0.078652

Property_Area	Loan_Status

Loan_Amount_Term	Loan_Status

Semiurban	0.768240
Urban	0.658416
Rural	0.614525

12	1.000000
120	1.000000
60	1.000000
240	0.750000
84	0.750000
360	0.701172
180	0.659091
300	0.615385
480	0.400000
36	0.000000

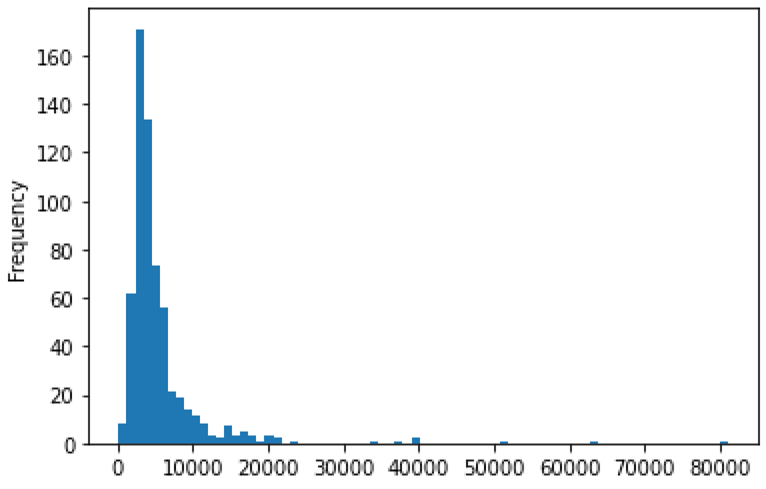
From the above tables, I found that values ‘Male’ and ‘Female’ in the Gender attribute have very similar values. Based on this, I think the Gender attribute does not affect the class label, and can be omitted from the models I’ll use. The same goes for Self Employed.

The most significant difference is in Credit History, where 79.58% of loans are accepted when Credit History is 1.0, and 7.87% when Credit History is 0.0. This shows that Credit History is the most important classifier when determining Loan_Status.

Loan Amount Term, despite being all numeric values, I felt was a nominal attribute as there are only 10 possible options. I decided that it was not a suitable feature column, as there seems to be no correlation between the length of the loan amount term and loan being accepted.

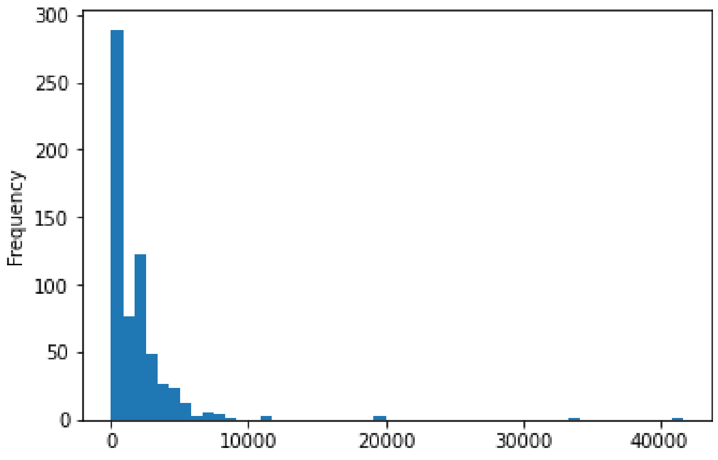
Analysis - Numeric Attributes

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
count	614.000000	614.000000	592.000000	600.00000
mean	5403.459283	1621.245798	146.412162	342.00000
std	6109.041673	2926.248369	85.587325	65.12041
min	150.000000	0.000000	9.000000	12.00000
25%	2877.500000	0.000000	100.000000	360.00000
50%	3812.500000	1188.500000	128.000000	360.00000
75%	5795.000000	2297.250000	168.000000	360.00000
Max	81000.000000	41667.000000	700.000000	480.00000

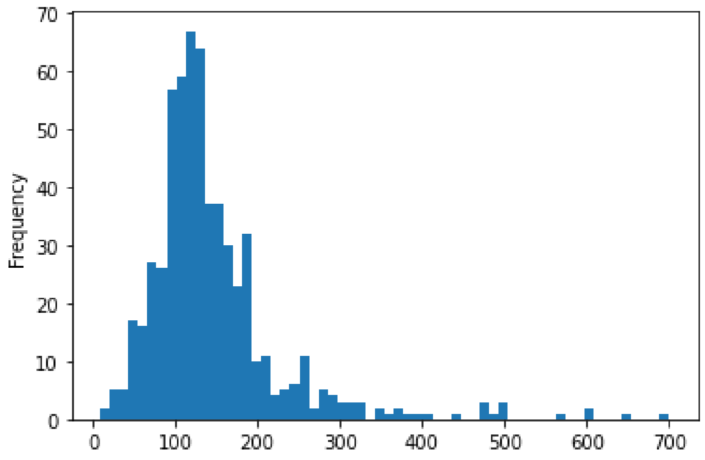


Applicant Income: The mean for Applicant Income is 5403, and the standard deviation is 6109. This shows there is a very high variance of the data. 75% of the data is below 5795, well below the mean. This means that the outliers (for example, at 81000) are artificially raising the mean, so the median would be a better indicator of centre. The median of Applicant Income is 3812.5. The interquartile range of the data is 2917.5, which shows that the majority of the data is positione

<u>CoapplicantIncome:</u> The mean for Co Applicant Income is 1621.25, with a standard deviation of 2926.25. Like Applicant Income. It is heavily influenced by extreme outliers like 41667, as the interquartile range is 2297.25. The median of Co Applicant Income is 1188.5, like Applicant Income, this is a better indicator of centre than the mean, as it is not influenced by outliers. The mode of the data is 0, as many of the



<u>Loan Amount:</u> The mean for Loan Amount is 146.4, with a standard deviation of 85.59. After looking at the distribution and standard deviation, it is clear that loan amount is more evenly distributed than Applicant and Co Applicant Income. The median is 128, which is much closer to the mean when compared to to Applicant and Co Applicant Income.



I decided to create bands for Applicant, Co Applicant Income and Loan Amount,with each band being of equal size. After grouping by those attributes, I found how tuples in each band were affected whether the loan got accepted or not.

ApplicantIncome	Loan_Status
(149.999, 3166.0]	0.695652
(3166.0, 4863.333]	0.683168
(4863.333, 81000.0]	0.682927

CoapplicantIncome	Loan_Status
(-0.001, 1188.5]	0.677524
(1188.5, 41667.0]	0.697068

LoanAmount	Loan_Status
(8.999, 110.0]	0.691542
(110.0, 151.0]	0.738462
(151.0, 700.0]	0.653061

The lower, middle and upper bands in Applicant Income are all accepted at similar rates. My prediction that Applicant Income heavily affects whether the loans getty accepted does not seem true in this dataset. In fact, the lowest band is accepted slightly more often than the upper band.

Co Applicant Income, like Applicant Income, does not drastically affect Loan Status, with the upper band slightly more likely to be accepted.

Loan Amount does seem to correlate with Loan Status, with the middle bracket of loans being the most likely to be accepted, and the upper band being the less likely to be accepted.

Algorithm:

K-Nearest-Neighbour

The attributes columns I used in my K-Nearest-Neighbour algorithm are

1. Married
2. Education
3. Co Applicant Income
4. Loan Amount
5. Credit History
6. Property Area

I assigned integer values to 0, 1, 2, 3 to Married, Education, Property Area attributes, and I decided to assign the band attributes to Applicant Income, Co Applicant Income and Loan Amount. I then assigned integer values to the bands. This, I felt, negated the need to normalise those attributes.

I split the data randomly between test and train datasets. When testing a points in the test dataset, I found the euclidean distance between that point and every other point in the train data. I then returned the class label of those closest points.

For example, u = (0, 1, 1, 3, 1, 6, 1) and v = (1, 1, 1, 1, 1, 6, 1).

v is my test point, and u is in the train dataset.

$$\sqrt{(0-1)^2+(1-1)^2+(1-1)^2+(3-1)^2+(1-1)^2+(6-6)^2+(1-1)^2}$$

$$= \sqrt{5}$$

If this was one of the closest 5 points in the train dataset, the class label (ie 'Y' or 'N') would be one of the 5 class labels.

Decision-Tree

The attributes I used in my Decision Tree were;

1. Married
2. Education
3. Co Applicant Income
4. Credit History
5. Property Area

Firstly, I decided to select the Attributes to split via Information Gain.

The entropy of the class 'Loan_Status' can be found as follows;

$-(P_{Yes} \log_2(P_{Yes}) + P_{No} (\log_2 P_{No}))$ $P_{Yes} = 422/614$ $P_{No} = 192 / 614$

$= - ((422 / 614 * (\log_2 422 / 614)) + (192 / 614 * (\log_2 192 / 614)))$

$= 0.896$

To find the Information Gain of Education for example, I would have to find the expected information required to classify a tuple after partitioning by Education;

$480 / 614 * (- 400 / 480 \log_2 (400 / 480) - 80 / 480 \log_2 (80 / 480))$

$+ \quad 134 / 614 * (- 82 / 134 \log_2 (82 / 134) - 52 / 134 \log_2 (52 / 134))$

$= (0.508) + (0.210) = 0.718$

So the Information Gain for Education is;

$0.896 - 0.718 = 0.178$ bits

I repeated this for all other attributes and found Credit History had the highest information Gain. Therefore, the first split was on Credit History.

This process is repeated on the sub datasets until either;

1. The class of all tuples remaining are all the same on a node
2. All attributes have been used, and the leaf returns the class label with a majority.

Results

I split my dataset into 4 dataframes,

1. X_train - contains feature attributes
2. Y_train - contains the class label for X_train's tuples
3. X_test - Contains feature tuples to test my algorithm on
4. Y_test - contains the actual class label of X_test

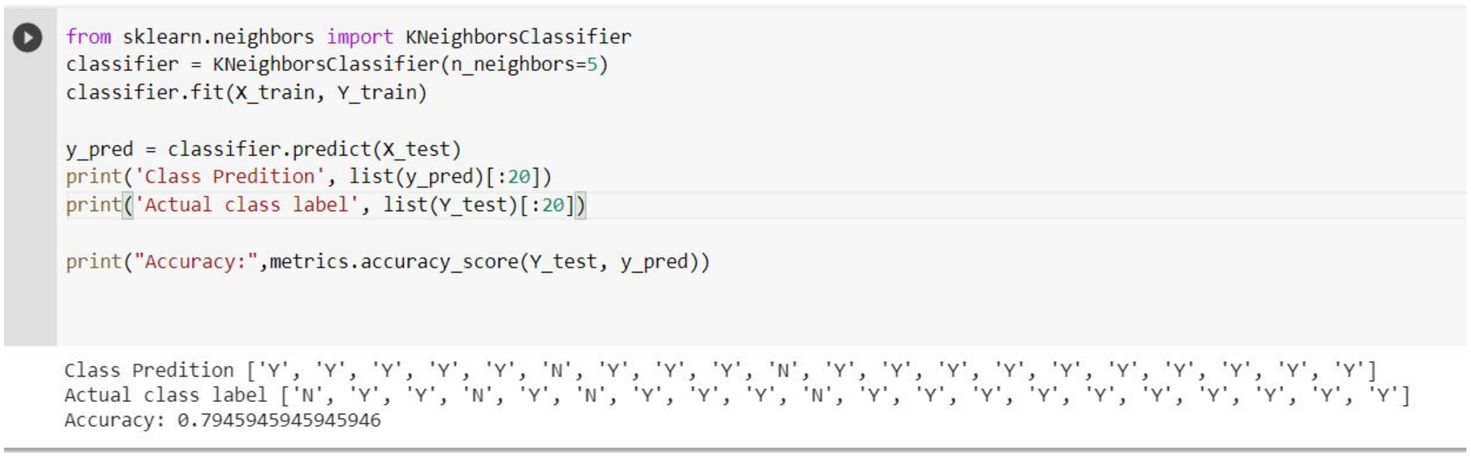
K-Nearest_Kneighbour

After training my algorithm using X_train and Y_train, I tested my algorithm with X_test. The result was a list of strings with either 'Y' or 'N'.

The algorithm had an accuracy of 79.46%.

I believe this could be improved upon by giving more appropriate weightings to every attribute, with 'Credit History' and 'Education' having the highest weightings and 'Co Applicant Income' having lower weightings.

A disadvantage of KNN is it does not work well for large dimensional data, so I tried to limit the amount of attributes to use.



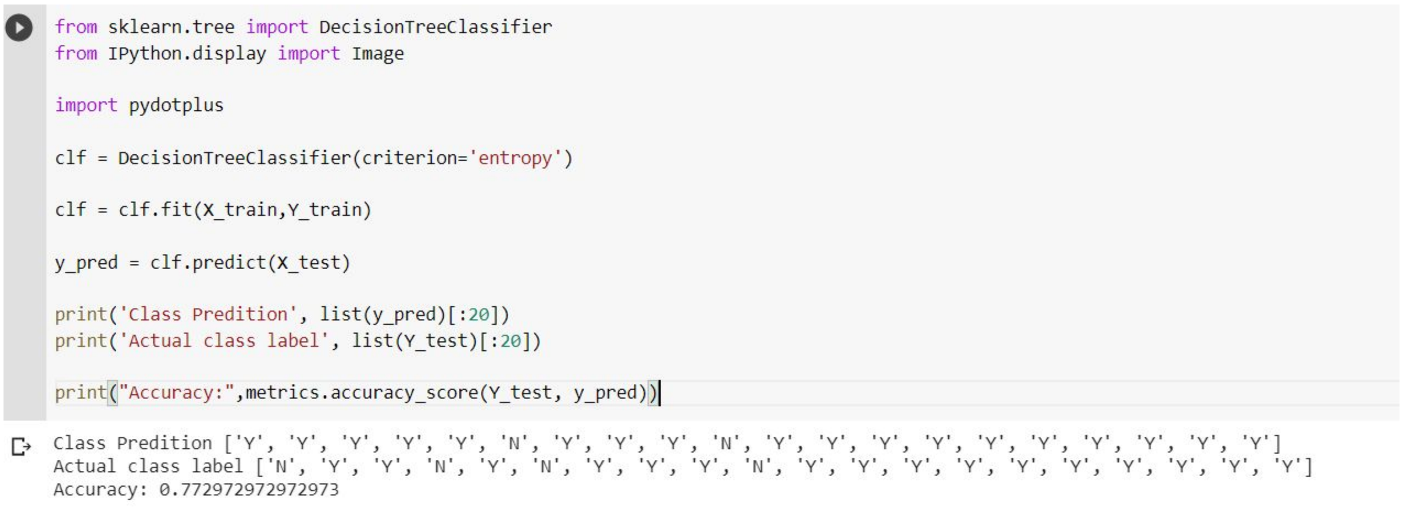
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, Y_train)

y_pred = classifier.predict(X_test)
print('Class Predition', list(y_pred)[:20])
print('Actual class label', list(Y_test)[:20])

print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Class Predition ['Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
Actual class label ['N', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
Accuracy: 0.7945945945945946
```

.

Decision Tree

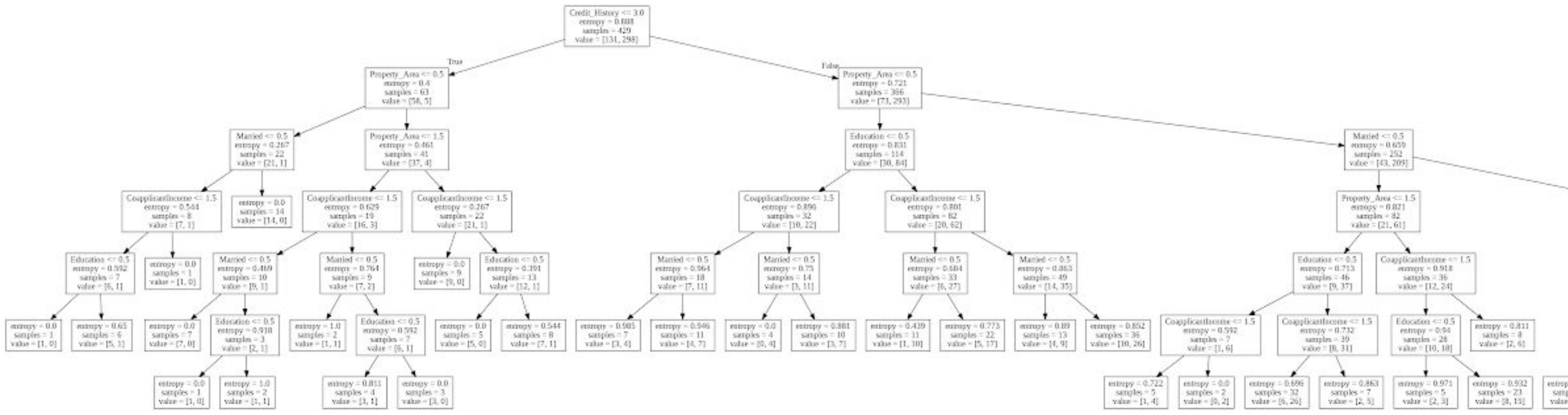
My decision tree algorithm gave an accuracy 77.3%.



```
from sklearn.tree import DecisionTreeClassifier
from IPython.display import Image

import pydotplus

clf = DecisionTreeClassifier(criterion='entropy')

clf = clf.fit(X_train,Y_train)

y_pred = clf.predict(X_test)

print('Class Predition', list(y_pred)[:20])
print('Actual class label', list(Y_test)[:20])

print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Class Predition ['Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
Actual class label ['N', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
Accuracy: 0.772972972972973
```

The decision tree looked like this,



There is still high entropy on many leaf nodes, which shows that attributes I selected may not have been the most appropriate, and using more or all attributes might have produced better results.

Conclusion of results

K Nearest Neighbour gave slightly more accurate results overall than Decision Tree algorithm. Problems which may have aggravated the accuracy of my algorithms could be where I split the numeric attributes, and missing data in certain tuples

Ultimately, the idea that looking at attributes such as ‘Credit History’ and ‘Married’ can definitely be used to predict whether a loan application is accepted or not.