# UO Predictive Analytics

## 1 Bayesian Classification

This notebook is a demonstration of Naive Bayes classifier development for a loan approval predici-ton problem. We apply the concepts discussed: Bayes theorem and The Naive Bayes classifier

We will use the following python libraries for this practical. - Pandas: https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html - scikit-learn: https://scikit-learn.org/stable/index.html

Additionally, we will use the mixed-naive-bayes 0.0.1 library available at https://pypi.org/project/mixed-naive-bayes/.

***** To install the package mixed-naive-bayes via pip in cmd.exe using the following**:

*pip install git+https://github.com/remykarem/mixed-naive-bayes#egg=mixed_naive_bayes*

If you are using Anaconda, please locate your Anaconda directory and use the command *activate* first.

*****For Mac users, please install Anaconda3 and use the following command in Terminal:**

*pip install mixed_naive_bayes*

Similar to last week's practical, we will follow the workflow discussed in Concept 1.2 Predictive analytics workflow.

### 1.1 Predict Loan Eligibility for Dream Housing Finance company

source: https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/#ProblemStatement

The dataset comes in csv format. Each sample has the following attributes:

| Variable | Description |
|---|---|
| Loan_ID | Unique Loan ID |
| Gender Married | Male/ Female |
| Dependents | Applicant married (Y/N) |
| Education | Number of dependents |
| Self_Employed | Applicant Education (Graduate/ Under Graduate) Self |
| ApplicantIncome | employed (Y/N) |
| CoapplicantIncome | Applicant income |
| LoanAmount | Coapplicant income |
| Loan_Amount_Term | Loan amount in thousands |
| Credit_History | Term of loan in months |
| Property_Area | credit history meets guidelines |
| Loan_Status | Urban/ Semi Urban/ Rural |
| | (Target) Loan approved (Y/N) |

Our goal is to create a Naive Bayes model that can learn from the training samples, so that we can predict outcome of a loan application.

[4]:
```
#imprt the required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
```

**If you do not remember the purposes of the above imported modules. Please revisit week 1 practicals.**

### 1.2 Step 1: Preprocessing

First, we load "loan_prediction" dataset from the CSV file as pandas dataframe and observe first five instances.

[6]:
```
df = pd.read_csv("loan_prediction.csv")
df.head(5)
```

[6]:
```
   Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001003   Male     Yes          1      Graduate            No
1  LP001005   Male     Yes          0      Graduate           Yes
2  LP001006   Male     Yes          0  Not Graduate            No
3  LP001008   Male      No          0      Graduate            No
4  LP001011   Male     Yes          2      Graduate           Yes

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4583             1508.0       128.0             360.0
1             3000                0.0        66.0             360.0
2             2583             2358.0       120.0             360.0
3             6000                0.0       141.0             360.0
4             5417             4196.0       267.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Rural           N
1             1.0         Urban           Y
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

[7]:
```
#inspect the values in the categorical features
df['Gender'].value_counts()
```

[7]:
```
Male      394
Female     86
Name: Gender, dtype: int64
```

[8]:
```
df['Married'].value_counts()
```

[8]:
```
Yes    311
No     169
Name: Married, dtype: int64
```

[9]:
```
df['Dependents'].value_counts()
```

[9]:
```
0     274
2      85
1      80
3+     41
Name: Dependents, dtype: int64
```

Dependents is a categorical feature as 3+ is not a number. We should use it as a categorical feature.

[10]:
```
df['Education'].value_counts()
```

[10]:
```
Graduate        383
Not Graduate     97
Name: Education, dtype: int64
```

[11]:
```
df['Self_Employed'].value_counts()
```

[11]:
```
No     414
Yes     66
Name: Self_Employed, dtype: int64
```

[12]:
```
df['Property_Area'].value_counts()
```

[12]:
```
Semiurban    191
Urban        150
Rural        139
Name: Property_Area, dtype: int64
```

Lets, take a quick look at the shape and summary of the dataset.

[13]:
```
#shape
df.shape
```

[13]:
```
(480, 13)
```

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            480 non-null    object
```

```
 1   Gender             480 non-null    object
 2   Married            480 non-null    object
 3   Dependents         480 non-null    object
 4   Education          480 non-null    object
 5   Self_Employed      480 non-null    object
 6   ApplicantIncome    480 non-null    int64
 7   CoapplicantIncome  480 non-null    float64
 8   LoanAmount         480 non-null    float64
 9   Loan_Amount_Term   480 non-null    float64
 10  Credit_History     480 non-null    float64
 11  Property_Area      480 non-null    object
 12  Loan_Status        480 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 48.9+ KB
```

We get three important information form shape and summary: 1. There are 480 instances and 12 attributes. The target is Loan_status. 1. From the Non-Null Count, we find that there is no missing values. Missing values affect the models adversely. We will learn about the effect of missing values and how to handle them later in the course. 1. dtypes at the bottom of the summary information tells us there are 4 floating point attributes, 1 integer attribute and 8 object or string valued attributes.

Similar to practical 1, we need to encode the categorical attributes.

[15]:
```
#load the library for encoding
from sklearn import preprocessing

le = preprocessing.LabelEncoder()

df['en_gender']        = le.fit_transform(df['Gender'] )
df['en_married']       = le.fit_transform(df['Married'] )
df['en_dependents']    = le.fit_transform(df['Dependents'] )
df['en_education']     = le.fit_transform(df['Education'] )
df['en_self_employed'] = le.fit_transform(df['Self_Employed'] )
df['en_parea']         = le.fit_transform(df['Property_Area'] )

#encoding the target
df['target']           = le.fit_transform(df['Loan_Status'] )
```

[16]:
```
#list the features
features = list(df.columns)
features
```

[16]:
```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
```

```
 'Self_Employed',
 'ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'Loan_Status',
 'en_gender',
 'en_married',
 'en_dependents',
 'en_education',
 'en_self_employed',
 'en_parea',
 'target']
```

We need to select the encoded features for our models. Also, laon id is an identifier i.e., it is different for each sample. We will exclude that from our model training.

[17]:
```
#remove laon id and target - Loan_Status from features
features.remove('Loan_ID')
features.remove('Loan_Status')
features.remove('target')

#remove the non encoded features from the feautre list
features.remove('Gender')
features.remove('Married')
features.remove('Dependents')
features.remove('Education')
features.remove('Self_Employed')
features.remove('Property_Area')

features
```

[17]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History',
 'en_gender',
 'en_married',
 'en_dependents',
 'en_education',
 'en_self_employed',
 'en_parea']
```

[18]:
```
#making sure we have 11 feautres in the list
len(features)
```

[18]:
```
11
```

Now, we check whether we have equal number of samples for each target class. A dataset where the samples are equally distributed to the target classes is called a balanced dataset. Imbalanced datasets are not good for training models as the model fails to perform well on samples who has the samples in the training set.

[19]:
```
df['Loan_Status'].value_counts()
```

[19]:
```
Y    332
N    148
Name: Loan_Status, dtype: int64
```

We observe that our data is not balanced. We have more samples where target class is Y than samples of class N.

Now we will divide our dataset into training and test sets. We will use a 70/30 split. Read about spliting using sklearn https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

As our dataset is imbalanced, we must use stratified sampling to split the dataset to ensure representative samples from all the target classes.

See the advantages of stratified sampling here https://en.wikipedia.org/wiki/Stratified_sampling.

[20]:
```
#stratified sampling
X_train, X_test, y_train, y_test = train_test_split(df[features],       #the␣
 ↪feature space
                                                    df['target'],       #target␣
 ↪column
                                                    test_size=0.30,     #25% in␣
 ↪the test set
                                                    random_state=412,
                                                    stratify = df['target']␣
 ↪#the feautre to stratify
                                                   )
```

[21]:
```
#check the class distribution in the test set
y_test.value_counts()
```

[21]:
```
1    100
0     44
Name: target, dtype: int64
```

## 1.3 Step 2: Learning the Naive Bayes model

To build a Naive Bayes classifier, w e c an u se t he i mpelementatio o f t he s klear p ackage. There are five d ifferent me thods ar e pr ovided fo r pr obability es timation. Th ey ar e: 1. GaussianNB - for continuous or numeric attribute 1. MultinomialNB - for nominal or categorical attribute 1. ComplementNB - improved method for nominal or categorical attribute 1. BernoulliNB - for binary attributes 1. CategoricalNB - for nominal or categorical attribute

If our dataset has one type of features only i.e., datatypes of the features are only numeric or nominal then we can use one of the above implementations to build a predictive model. However, the features in the loan_prediction dataset are of mixed types. Here, we cannot use sklearn.

The mixed-naive-bayes 0.0.1 (https://pypi.org/project/mixed-naive-bayes/#quick-start) provides an implementation of Naive Bayes for mixed attributes. We will use this python library for this practical.

Note: The module expects that we have label encoded the categorical features.

[22]:
```
#initialize the mixed_naive_bayes lib
from mixed_naive_bayes import MixedNB

#fit the model with the training set
clfNB = MixedNB(categorical_features=[5,6,7,8,9,10])
#categorical_features is a list of indices categorical attributes in our dataset

clfNB.fit(X_train, y_train)
```

[22]:
```
MixedNB(alpha=0.5, var_smoothing=1e-09)
```

## 1.4 Step 3: Evaluation

We have trained our model in the previous step which is represented as knn. Read about sklearn kNN method here https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

We will now evalute the performance of our model on the test set. That is, we will apply the model to the test set X_test and match the predictions of the model with y_test.

[23]:
```
#predictions
y_pred = clfNB.predict(X_test)
y_pred
```

[23]:
```
array([1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1], dtype=int64)
```

[24]:
```
#here is our true labels
y_test
```

[24]:
```
421    1
394    1
347    1
317    1
454    0
      ..
17     1
31     1
72     1
445    0
307    1
Name: target, Length: 144, dtype: int32
```

Detailed list of parameters of MixedNB is provided here https://remykarem.github.io/docs/mixed_naive_bayes.html

For the test instances, we can inspect the class probabilities as well.

[25]:
```
#let us take the first test instance for example.
X_test[0:1]
```

[25]:
```
     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
421             6417                0.0       157.0             180.0

     Credit_History  en_gender  en_married  en_dependents  en_education  \
421             1.0          1           1              3             0

     en_self_employed  en_parea
421                 0         0
```

[26]:
```
clfNB.predict_proba(X_test[0:1])
```

[26]:
```
array([[1.91465654e-11, 2.81453864e-11]])
```

### 1.4.1 Explanation

Given an instance $\mathbf{x}' = (6417, 0.0, 157.0, 180.0, 1.0, 1, 1, 3, 0, 0, 0)$, the model predicts

$P(Y|\mathbf{x}') = 1.915 \times 10^{-11}$

$P(N|\mathbf{x}') = 2.815 \times 10^{-11}$

We can say that the model predicted that the application is not eligible for a loan.

[27]:
```
#measure accuracy
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred)
```

[27]: 0.8125

[28]:
```
#we can also use the score method of MixedNB to observe the accuracy
clfNB.score(X_test, y_test)
```

[28]: 0.8125

Our Naive Bayes model has 81.25% accuracy on the loan_prediction dataset.
In this practical, we work through all the steps required to develop a Naive Bayes model. We used a 70/30 splits for taining and test sets.