In this notebook we try to practice all the classification algorithms that we have learned in this course.

We load a dataset using Pandas library, and apply the following algorithms, and find the best one for this specific dataset by accuracy evaluation methods.

Let's first load required libraries:

In [ ]:

**import** itertools  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** matplotlib.ticker **import** NullFormatter  
**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib.ticker **as** ticker  
**from** sklearn **import** preprocessing  
**%matplotlib** inline

About dataset[¶](#gjdgxs)

This dataset is about past loans. The **Loan\_train.csv** data set includes details of 346 customers whose loan are already paid off or defaulted. It includes following fields:

| **Field** | **Description** |
| --- | --- |
| Loan\_status | Whether a loan is paid off on in collection |
| Principal | Basic principal loan amount at the |
| Terms | Origination terms which can be weekly (7 days), biweekly, and monthly payoff schedule |
| Effective\_date | When the loan got originated and took effects |
| Due\_date | Since it’s one-time payoff schedule, each loan has one single due date |
| Age | Age of applicant |
| Education | Education of applicant |
| Gender | The gender of applicant |

Let's download the dataset

In [ ]:

**!**wget -O loan\_train.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule\_Coursera/data/loan\_train.csv

"wget" no se reconoce como un comando interno o externo,  
programa o archivo por lotes ejecutable.

Load Data From CSV File[¶](#30j0zll)

In [ ]:

**import** os

In [ ]:

filename **=** 'loan\_train\_test.csv'  
df **=** pd**.**read\_csv(filename)

In [ ]:

df**.**shape

Out[ ]:

(346, 11)

Convert to date time object[¶](#1fob9te)

In [ ]:

df['due\_date'] **=** pd**.**to\_datetime(df['due\_date'])  
df['effective\_date'] **=** pd**.**to\_datetime(df['effective\_date'])  
df**.**head()

Out[ ]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **Unnamed: 0.1.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | male |
| **1** | 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | female |
| **2** | 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | male |
| **3** | 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | female |
| **4** | 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | male |

Data visualization and pre-processing[¶](#3znysh7)

Let’s see how many of each class is in our data set

In [ ]:

df['loan\_status']**.**value\_counts()

Out[ ]:

PAIDOFF 260  
COLLECTION 86  
Name: loan\_status, dtype: int64

260 people have paid off the loan on time while 86 have gone into collection

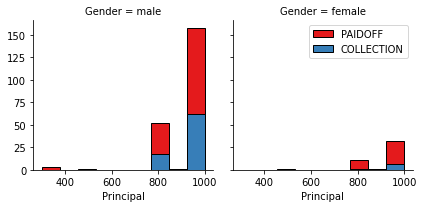
Let's plot some columns to underestand data better:

In [ ]:

*# notice: installing seaborn might takes a few minutes*  
*#!conda install -c anaconda seaborn -y*

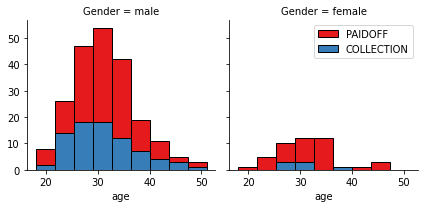
In [ ]:

**import** seaborn **as** sns  
  
bins **=** np**.**linspace(df**.**Principal**.**min(), df**.**Principal**.**max(), 10)  
g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)  
g**.**map(plt**.**hist, 'Principal', bins**=**bins, ec**=**"k")  
  
g**.**axes[**-**1]**.**legend()  
plt**.**show()



In [ ]:

bins **=** np**.**linspace(df**.**age**.**min(), df**.**age**.**max(), 10)  
g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)  
g**.**map(plt**.**hist, 'age', bins**=**bins, ec**=**"k")  
  
g**.**axes[**-**1]**.**legend()  
plt**.**show()

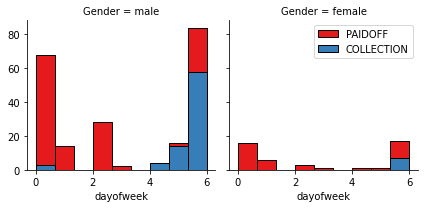


Pre-processing: Feature selection/extraction[¶](#2et92p0)

Let's look at the day of the week people get the loan[¶](#tyjcwt)

In [ ]:

df['dayofweek'] **=** df['effective\_date']**.**dt**.**dayofweek  
bins **=** np**.**linspace(df**.**dayofweek**.**min(), df**.**dayofweek**.**max(), 10)  
g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)  
g**.**map(plt**.**hist, 'dayofweek', bins**=**bins, ec**=**"k")  
g**.**axes[**-**1]**.**legend()  
plt**.**show()



We see that people who get the loan at the end of the week don't pay it off, so let's use Feature binarization to set a threshold value less than day 4

In [ ]:

df['weekend'] **=** df['dayofweek']**.**apply(**lambda** x: 1 **if** (x**>**3) **else** 0)  
df**.**head()

Out[ ]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **Unnamed: 0.1.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** | **dayofweek** | **weekend** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | male | 3 | 0 |
| **1** | 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | female | 3 | 0 |
| **2** | 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | male | 3 | 0 |
| **3** | 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | female | 4 | 1 |
| **4** | 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | male | 4 | 1 |

Convert Categorical features to numerical values[¶](#3dy6vkm)

Let's look at gender:

In [ ]:

df**.**groupby(['Gender'])['loan\_status']**.**value\_counts(normalize**=True**)

Out[ ]:

Gender loan\_status  
female PAIDOFF 0.865385  
 COLLECTION 0.134615  
male PAIDOFF 0.731293  
 COLLECTION 0.268707  
Name: loan\_status, dtype: float64

86 % of female pay there loans while only 73 % of males pay there loan

Let's convert male to 0 and female to 1:

In [ ]:

df['Gender']**.**replace(to\_replace**=**['male','female'], value**=**[0,1],inplace**=True**)  
df**.**head()

Out[ ]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **Unnamed: 0.1.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** | **dayofweek** | **weekend** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | 0 | 3 | 0 |
| **1** | 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | 1 | 3 | 0 |
| **2** | 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | 0 | 3 | 0 |
| **3** | 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | 1 | 4 | 1 |
| **4** | 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | 0 | 4 | 1 |

One Hot Encoding[¶](#1t3h5sf)

#### How about education?[¶](#4d34og8)

In [ ]:

df**.**groupby(['education'])['loan\_status']**.**value\_counts(normalize**=True**)

Out[ ]:

education loan\_status  
Bechalor PAIDOFF 0.750000  
 COLLECTION 0.250000  
High School or Below PAIDOFF 0.741722  
 COLLECTION 0.258278  
Master or Above COLLECTION 0.500000  
 PAIDOFF 0.500000  
college PAIDOFF 0.765101  
 COLLECTION 0.234899  
Name: loan\_status, dtype: float64

Features before One Hot Encoding[¶](#2s8eyo1)

In [ ]:

df[['Principal','terms','age','Gender','education']]**.**head()

Out[ ]:

|  | **Principal** | **terms** | **age** | **Gender** | **education** |
| --- | --- | --- | --- | --- | --- |
| **0** | 1000 | 30 | 45 | 0 | High School or Below |
| **1** | 1000 | 30 | 33 | 1 | Bechalor |
| **2** | 1000 | 15 | 27 | 0 | college |
| **3** | 1000 | 30 | 28 | 1 | college |
| **4** | 1000 | 30 | 29 | 0 | college |

Use one hot encoding technique to conver categorical varables to binary variables and append them to the feature Data Frame[¶](#17dp8vu)

In [ ]:

Feature **=** df[['Principal','terms','age','Gender','weekend']]  
Feature **=** pd**.**concat([Feature,pd**.**get\_dummies(df['education'])], axis**=**1)  
Feature**.**drop(['Master or Above'], axis **=** 1,inplace**=True**)  
Feature**.**head()

Out[ ]:

|  | **Principal** | **terms** | **age** | **Gender** | **weekend** | **Bechalor** | **High School or Below** | **college** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1000 | 30 | 45 | 0 | 0 | 0 | 1 | 0 |
| **1** | 1000 | 30 | 33 | 1 | 0 | 1 | 0 | 0 |
| **2** | 1000 | 15 | 27 | 0 | 0 | 0 | 0 | 1 |
| **3** | 1000 | 30 | 28 | 1 | 1 | 0 | 0 | 1 |
| **4** | 1000 | 30 | 29 | 0 | 1 | 0 | 0 | 1 |

Feature Selection[¶](#3rdcrjn)

Let's define feature sets, X:

In [ ]:

X **=** Feature  
X[0:5]

Out[ ]:

|  | **Principal** | **terms** | **age** | **Gender** | **weekend** | **Bechalor** | **High School or Below** | **college** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1000 | 30 | 45 | 0 | 0 | 0 | 1 | 0 |
| **1** | 1000 | 30 | 33 | 1 | 0 | 1 | 0 | 0 |
| **2** | 1000 | 15 | 27 | 0 | 0 | 0 | 0 | 1 |
| **3** | 1000 | 30 | 28 | 1 | 1 | 0 | 0 | 1 |
| **4** | 1000 | 30 | 29 | 0 | 1 | 0 | 0 | 1 |

What are our lables?

In [ ]:

y **=** df['loan\_status']**.**values  
y[0:5]

Out[ ]:

array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],  
 dtype=object)

Normalize Data[¶](#26in1rg)

Data Standardization give data zero mean and unit variance (technically should be done after train test split)

In [ ]:

X**=** preprocessing**.**StandardScaler()**.**fit(X)**.**transform(X)  
X[0:5]

Out[ ]:

array([[ 0.51578458, 0.92071769, 2.33152555, -0.42056004, -1.20577805,  
 -0.38170062, 1.13639374, -0.86968108],  
 [ 0.51578458, 0.92071769, 0.34170148, 2.37778177, -1.20577805,  
 2.61985426, -0.87997669, -0.86968108],  
 [ 0.51578458, -0.95911111, -0.65321055, -0.42056004, -1.20577805,  
 -0.38170062, -0.87997669, 1.14984679],  
 [ 0.51578458, 0.92071769, -0.48739188, 2.37778177, 0.82934003,  
 -0.38170062, -0.87997669, 1.14984679],  
 [ 0.51578458, 0.92071769, -0.3215732 , -0.42056004, 0.82934003,  
 -0.38170062, -0.87997669, 1.14984679]])

Classification[¶](#lnxbz9)

Now, it is your turn, use the training set to build an accurate model. Then use the test set to report the accuracy of the model You should use the following algorithm:

* K Nearest Neighbor(KNN)
* Decision Tree
* Support Vector Machine
* Logistic Regression

\_\_ Notice:\_\_

* You can go above and change the pre-processing, feature selection, feature-extraction, and so on, to make a better model.
* You should use either scikit-learn, Scipy or Numpy libraries for developing the classification algorithms.
* You should include the code of the algorithm in the following cells.

K Nearest Neighbor(KNN)[¶](#35nkun2)

Notice: You should find the best k to build the model with the best accuracy.\ **warning:** You should not use the **loan\_test.csv** for finding the best k, however, you can split your train\_loan.csv into train and test to find the best **k**.

In [ ]:

**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.metrics **import** jaccard\_score  
**from** sklearn.metrics **import** f1\_score  
**from** sklearn.metrics **import** log\_loss

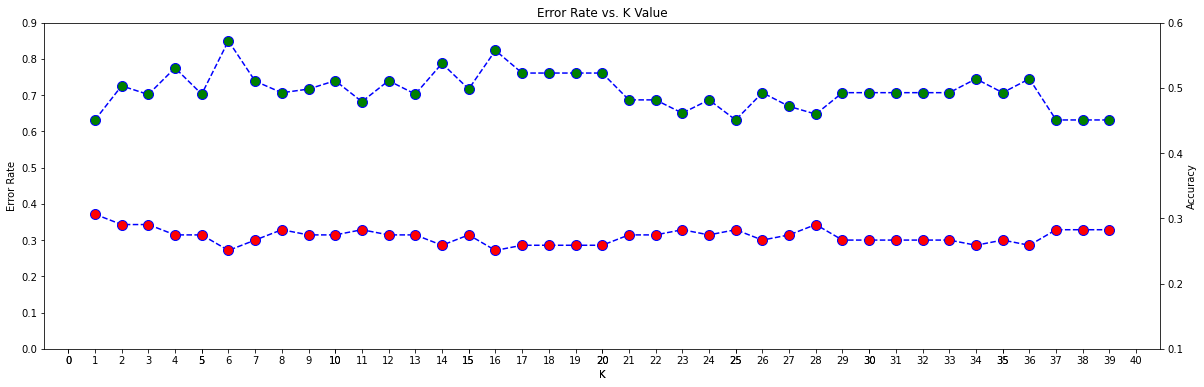
In [ ]:

Xtrain,Xtest, ytrain,ytest **=** train\_test\_split(X,y,test\_size**=**0.2,random\_state**=**7)

In [ ]:

error\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 error\_rate**.**append(np**.**mean(pred\_i **!=** ytest))  
  
acc\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 acc\_rate**.**append(jaccard\_score(ytest,pred\_i,average**=**'weighted'))  
  
fig **=** plt**.**figure(figsize**=**(20,6))  
*#plt.figure(figsize=(20,6))*  
ax **=** fig**.**add\_subplot(1,1,1,label**=**'1')  
ax1 **=** fig**.**add\_subplot(1,1,1,label**=**'2',frame\_on**=False**)  
ax**.**plot(range(1,40),error\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'red', markersize**=**10)  
ax**.**set\_title('Error Rate vs. K Value')  
ax**.**set\_xticks(np**.**arange(0,40,1))  
ax**.**set\_yticks(np**.**arange(0,1,0.1))  
ax**.**set\_xlabel('K')  
ax**.**set\_ylabel('Error Rate')  
  
ax1**.**plot(range(1,40),acc\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
*#ax1.set\_title('Accuracy vs. K Value')*  
ax1**.**set\_yticks(np**.**arange(0.1,0.7,0.1))  
*#ax1.set\_xticks(np.arange(0,40,1))*  
ax1**.**set\_xlabel('K')  
ax1**.**set\_ylabel('Accuracy')  
ax1**.**yaxis**.**tick\_right()  
ax1**.**yaxis**.**set\_label\_position('right')  
print("Minimum error: ",min(error\_rate),"at K =",error\_rate**.**index(min(error\_rate))**+**1)  
print("Maximun Jaccard Score: ",max(acc\_rate),"at K =",acc\_rate**.**index(max(acc\_rate))**+**1)

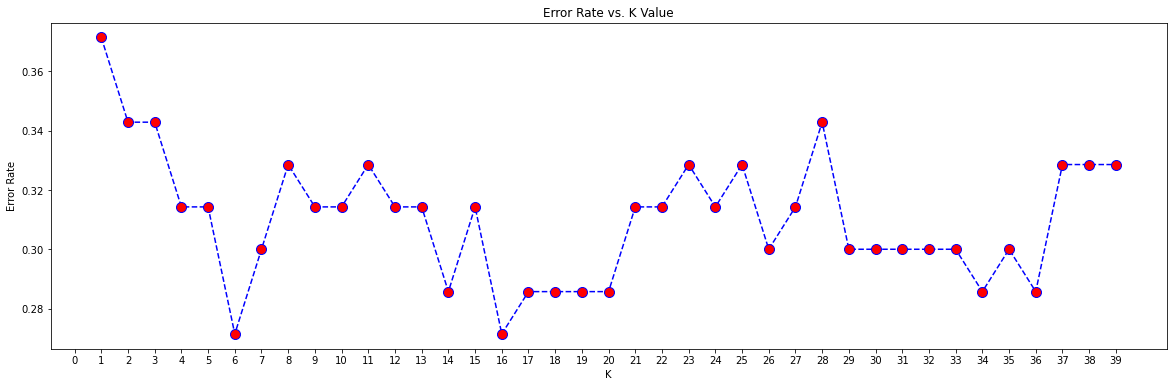
Minimum error: 0.2714285714285714 at K = 6  
Maximun Jaccard Score: 0.5721100164203612 at K = 6



In [ ]:

error\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 error\_rate**.**append(np**.**mean(pred\_i **!=** ytest))  
  
plt**.**figure(figsize**=**(20,6))  
plt**.**plot(range(1,40),error\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'red', markersize**=**10)  
plt**.**title('Error Rate vs. K Value')  
plt**.**xticks(np**.**arange(0,40,1))  
plt**.**xlabel('K')  
plt**.**ylabel('Error Rate')  
  
print("Minimum error:-",min(error\_rate),"at K =",error\_rate**.**index(min(error\_rate))**+**1)

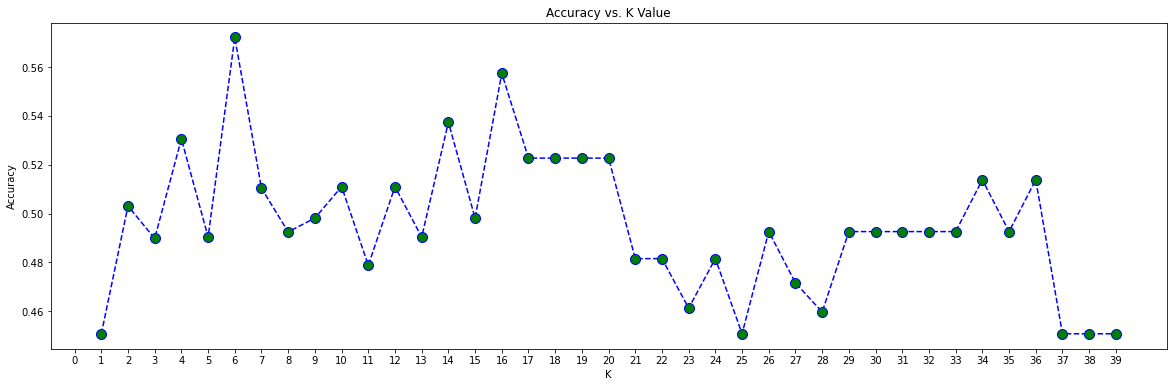
Minimum error:- 0.2714285714285714 at K = 6



In [ ]:

acc\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 acc\_rate**.**append(jaccard\_score(ytest,pred\_i,average**=**'weighted'))  
  
plt**.**figure(figsize**=**(20,6))  
plt**.**plot(range(1,40),acc\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
plt**.**title('Accuracy vs. K Value')  
plt**.**xticks(np**.**arange(0,40,1))  
plt**.**xlabel('K')  
plt**.**ylabel('Accuracy')  
print("Maximun Jaccard Score: ",max(acc\_rate),"at K =",acc\_rate**.**index(max(acc\_rate))**+**1)

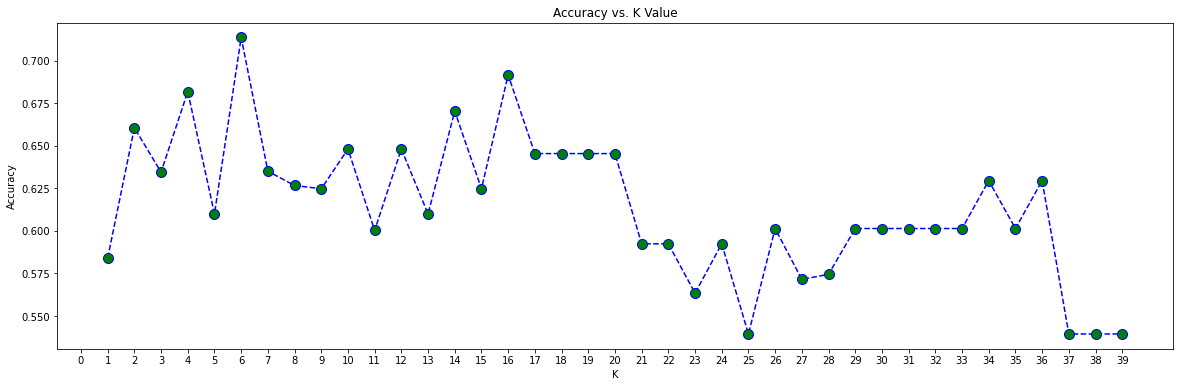
Maximun Jaccard Score: 0.5721100164203612 at K = 6



In [ ]:

acc\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 acc\_rate**.**append(f1\_score(ytest,pred\_i,average**=**'weighted'))  
  
plt**.**figure(figsize**=**(20,6))  
plt**.**plot(range(1,40),acc\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
plt**.**title('Accuracy vs. K Value')  
plt**.**xticks(np**.**arange(0,40,1))  
plt**.**xlabel('K')  
plt**.**ylabel('Accuracy')  
print("Maximun Jaccard Score: ",max(acc\_rate),"at K =",acc\_rate**.**index(max(acc\_rate))**+**1)

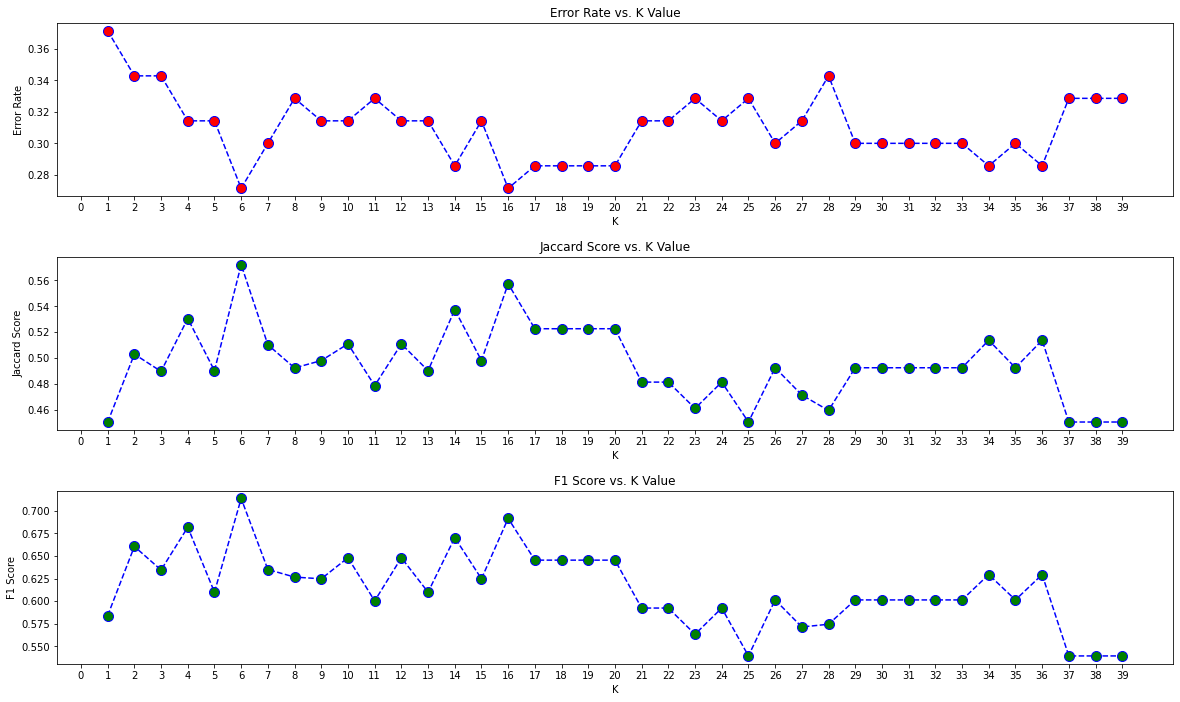
Maximun Jaccard Score: 0.7136183948065137 at K = 6



In [ ]:

error\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 error\_rate**.**append(np**.**mean(pred\_i **!=** ytest))  
  
jacc\_acc\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 jacc\_acc\_rate**.**append(jaccard\_score(ytest,pred\_i,average**=**'weighted'))  
  
f1\_acc\_rate **=** []  
**for** k **in** range(1,40):  
 knn **=** KNeighborsClassifier(n\_neighbors**=**k)  
 knn**.**fit(Xtrain,ytrain)  
 pred\_i **=** knn**.**predict(Xtest)  
 f1\_acc\_rate**.**append(f1\_score(ytest,pred\_i,average**=**'weighted'))  
  
*# Defino la estructura del grafico*  
fig **=** plt**.**figure(figsize**=**(20,10))  
ax **=** fig**.**add\_subplot(3,1,1)  
ax1 **=** fig**.**add\_subplot(3,1,2)  
ax2 **=** fig**.**add\_subplot(3,1,3)  
  
*# Defino el primer line plot*  
ax**.**plot(range(1,40),error\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'red', markersize**=**10)  
ax**.**set\_title('Error Rate vs. K Value')  
ax**.**set\_xticks(np**.**arange(0,40,1))  
ax**.**set\_xlabel('K')  
ax**.**set\_ylabel('Error Rate')  
  
*# Defino el segundo line plot*  
ax1**.**plot(range(1,40),jacc\_acc\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
ax1**.**set\_title('Jaccard Score vs. K Value')  
ax1**.**set\_xticks(np**.**arange(0,40,1))  
ax1**.**set\_xlabel('K')  
ax1**.**set\_ylabel('Jaccard Score')  
  
*# Defino el tercer line plot*  
*#ax2.figure(figsize=(20,6))*  
ax2**.**plot(range(1,40),f1\_acc\_rate,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
ax2**.**set\_title('F1 Score vs. K Value')  
ax2**.**set\_xticks(np**.**arange(0,40,1))  
ax2**.**set\_xlabel('K')  
ax2**.**set\_ylabel('F1 Score')  
  
*# Defino la distribucion para que el grafico no se superponga y sea mas armonico*  
plt**.**subplots\_adjust(left**=**0.125,  
 bottom**=**0.01,   
 right**=**0.9,   
 top**=**0.9,   
 wspace**=**0.2,   
 hspace**=**0.35)  
  
print("Minimum error: ",min(error\_rate),"at K =",error\_rate**.**index(min(error\_rate))**+**1)  
print("Maximun Jaccard Score: ",max(jacc\_acc\_rate),"at K =",jacc\_acc\_rate**.**index(max(jacc\_acc\_rate))**+**1)  
print("Maximun F1 Score: ",max(f1\_acc\_rate),"at K =",f1\_acc\_rate**.**index(max(f1\_acc\_rate))**+**1)

Minimum error: 0.2714285714285714 at K = 6  
Maximun Jaccard Score: 0.5721100164203612 at K = 6  
Maximun F1 Score: 0.7136183948065137 at K = 6



In [ ]:

result\_matriz **=** {'Algorithm':['KNN','Decision Tree','SVM','LogisticRegression'],  
'Jaccard':[0,0,0,0],'F1 score':[0,0,0,0],'LogLoss':['NA','NA','NA',0],}  
result\_matriz **=** pd**.**DataFrame(result\_matriz)  
print("Minimum error: ",min(error\_rate),"at K =",error\_rate**.**index(min(error\_rate))**+**1)  
print("Maximun Jaccard Score: ",max(jacc\_acc\_rate),"at K =",jacc\_acc\_rate**.**index(max(jacc\_acc\_rate))**+**1)  
print("Maximun F1 Score: ",max(f1\_acc\_rate),"at K =",f1\_acc\_rate**.**index(max(f1\_acc\_rate))**+**1)

Minimum error: 0.2714285714285714 at K = 6  
Maximun Jaccard Score: 0.5721100164203612 at K = 6  
Maximun F1 Score: 0.7136183948065137 at K = 6

Ambos algoritmos convergen en k=6 con Jaccard Score = 0,57 en test data y F1 Score = 0,71 en test data[¶](#1ksv4uv)

In [ ]:

result\_matriz**.**iloc[0,1] **=** max(jacc\_acc\_rate)  
result\_matriz**.**iloc[0,2] **=** max(f1\_acc\_rate)  
result\_matriz

Out[ ]:

|  | **Algorithm** | **Jaccard** | **F1 score** | **LogLoss** |
| --- | --- | --- | --- | --- |
| **0** | KNN | 0.57211 | 0.713618 | NA |
| **1** | Decision Tree | 0.00000 | 0.000000 | NA |
| **2** | SVM | 0.00000 | 0.000000 | NA |
| **3** | LogisticRegression | 0.00000 | 0.000000 | 0 |

Decision Tree[¶](#44sinio)

In [ ]:

**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn.metrics **import** jaccard\_score  
**from** sklearn.metrics **import** f1\_score  
**from** sklearn.metrics **import** log\_loss  
**from** io **import** StringIO  
**import** pydotplus  
**import** matplotlib.image **as** mpimg  
**from** sklearn **import** tree

In [ ]:

decision\_tree **=** DecisionTreeClassifier(criterion **=** 'entropy', max\_depth**=**3)  
decision\_tree**.**fit(Xtrain,ytrain)  
yhat **=** decision\_tree**.**predict(Xtest)

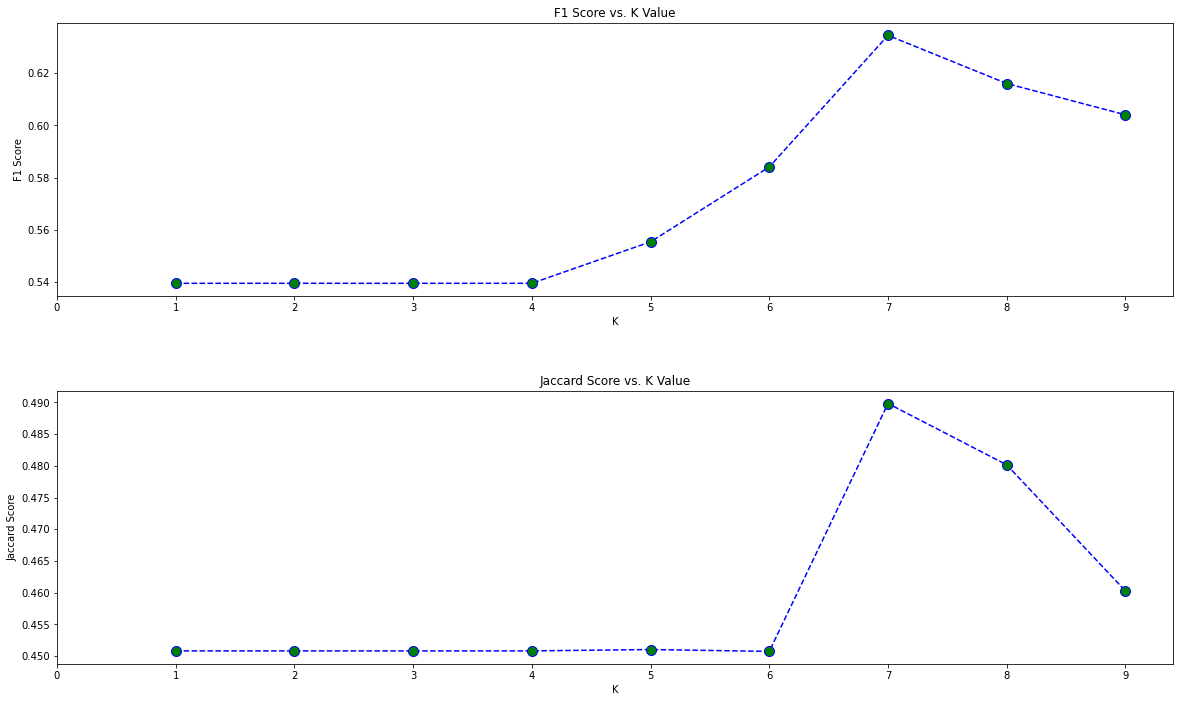
In [ ]:

jacc\_acc\_score **=** []  
**for** i **in** range(1,10):  
 decision\_tree **=** DecisionTreeClassifier(criterion**=**'entropy', max\_depth**=**i)  
 decision\_tree**.**fit(Xtrain,ytrain)  
 pred\_i **=** decision\_tree**.**predict(Xtest)  
 jacc\_acc\_score**.**append(jaccard\_score(ytest,pred\_i,average**=**'weighted'))  
  
f1\_acc\_score **=** []  
**for** i **in** range(1,10):  
 decision\_tree **=** DecisionTreeClassifier(criterion**=**'entropy', max\_depth**=**i)  
 decision\_tree**.**fit(Xtrain,ytrain)  
 pred\_i **=** decision\_tree**.**predict(Xtest)  
 f1\_acc\_score**.**append(f1\_score(ytest,pred\_i,average**=**'weighted'))  
  
print('Jaccard score: ',max(jacc\_acc\_score),  
 ' with depth = ',f1\_acc\_score**.**index(max(f1\_acc\_score))**+**1)  
print('F1 score: ',max(f1\_acc\_score),  
 ' with depth = ',jacc\_acc\_score**.**index(max(jacc\_acc\_score))**+**1)

Jaccard score: 0.48983980689049816 with depth = 7  
F1 score: 0.6344980097302078 with depth = 7

In [ ]:

*# Ploteo el grafico para analizar como convergen los algoritmos*  
*# Defino la estructura del grafico*  
fig **=** plt**.**figure(figsize**=**(20,10))  
ax **=** fig**.**add\_subplot(2,1,1)  
ax1 **=** fig**.**add\_subplot(2,1,2)  
  
*# Defino el primer line plot*  
ax**.**plot(range(1,10),f1\_acc\_score,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
ax**.**set\_title('F1 Score vs. K Value')  
ax**.**set\_xticks(np**.**arange(0,10,1))  
ax**.**set\_xlabel('K')  
ax**.**set\_ylabel('F1 Score')  
  
*# Defino el segundo line plot*  
ax1**.**plot(range(1,10),jacc\_acc\_score,color**=**'blue', linestyle**=**'dashed',   
 marker**=**'o',markerfacecolor**=**'green', markersize**=**10)  
ax1**.**set\_title('Jaccard Score vs. K Value')  
ax1**.**set\_xticks(np**.**arange(0,10,1))  
ax1**.**set\_xlabel('K')  
ax1**.**set\_ylabel('Jaccard Score')  
  
*# Defino la distribucion para que el grafico no se superponga y sea mas armonico*  
plt**.**subplots\_adjust(left**=**0.125,  
 bottom**=**0.01,   
 right**=**0.9,   
 top**=**0.9,   
 wspace**=**0.2,   
 hspace**=**0.35)

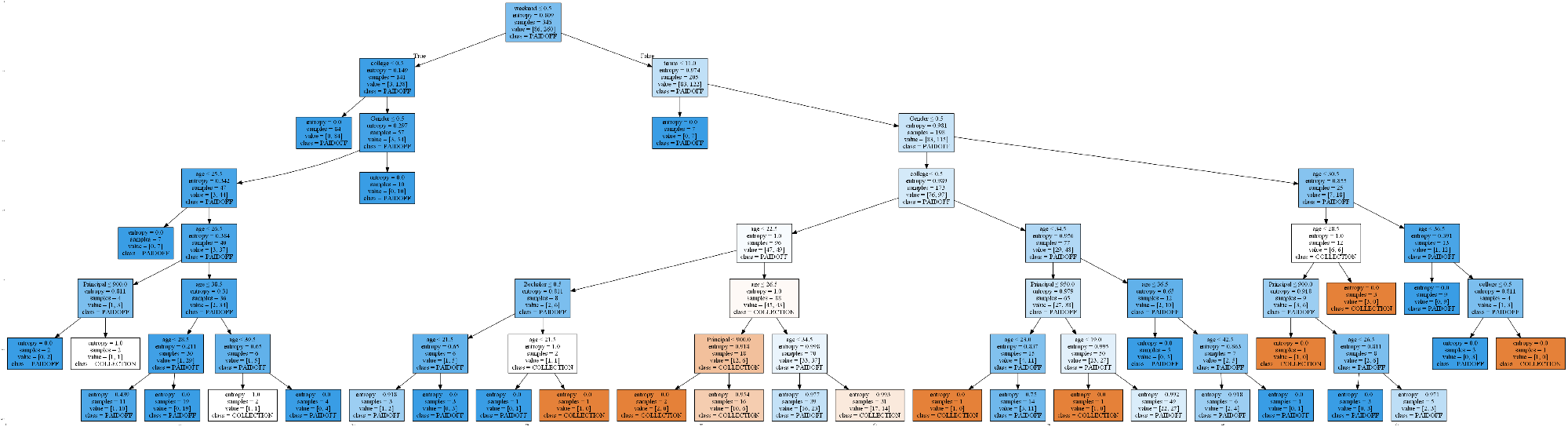


In [ ]:

*# Este paso no es necesario, solo lo hago para plotear el arbol*  
X\_tree **=** Feature**.**values  
y\_tree **=** df['loan\_status']**.**values  
decision\_tree **=** DecisionTreeClassifier(criterion**=**'entropy',max\_depth**=**7)  
decision\_tree**.**fit(X\_tree,y\_tree)  
dot\_data **=** StringIO()  
filename **=** "Loan\_class.png"  
featureNames **=** Feature**.**columns  
out**=**tree**.**export\_graphviz(decision\_tree,feature\_names**=**featureNames, out\_file**=**dot\_data, class\_names**=** np**.**unique(df['loan\_status']), filled**=True**, special\_characters**=True**,rotate**=False**)   
graph **=** pydotplus**.**graph\_from\_dot\_data(dot\_data**.**getvalue())   
graph**.**write\_png(filename)  
img **=** mpimg**.**imread(filename)  
plt**.**figure(figsize**=**(200, 200))  
plt**.**imshow(img,interpolation**=**'nearest')

Out[ ]:

<matplotlib.image.AxesImage at 0x2712c51fbb0>



In [ ]:

result\_matriz**.**iloc[1,1] **=** max(jacc\_acc\_score)  
result\_matriz**.**iloc[1,2] **=** max(f1\_acc\_score)  
result\_matriz

Out[ ]:

|  | **Algorithm** | **Jaccard** | **F1 score** | **LogLoss** |
| --- | --- | --- | --- | --- |
| **0** | KNN | 0.57211 | 0.713618 | NA |
| **1** | Decision Tree | 0.48984 | 0.634498 | NA |
| **2** | SVM | 0.00000 | 0.000000 | NA |
| **3** | LogisticRegression | 0.00000 | 0.000000 | 0 |

Support Vector Machine[¶](#2jxsxqh)

In [ ]:

**import** scipy.optimize **as** opt  
**from** sklearn **import** svm  
**from** sklearn.svm **import** SVC  
**from** sklearn.model\_selection **import** learning\_curve,GridSearchCV  
**from** sklearn.model\_selection **import** LeaveOneOut

SVM Parameter tunning[¶](#z337ya)

In [ ]:

**%%capture**  
*# Creo el modelo*  
SVM\_model **=** SVC()  
*# Inicializo los parametros para encontrar la mejor combinacion posible*  
param\_grid **=** {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel':['linear','rbf','poly','sigmoid']}  
linear\_model **=** GridSearchCV(SVM\_model,param\_grid,refit**=True**,verbose**=**2)  
linear\_model**.**fit(Xtrain,ytrain)

In [ ]:

print(linear\_model**.**best\_estimator\_)

SVC(C=0.1, gamma=1, kernel='linear')

In [ ]:

c **=** 0.1  
gamma **=** 1  
kernel **=** 'linear'  
  
SVM\_model **=** SVC(kernel **=** kernel,C**=**c , gamma**=**gamma)  
SVM\_model**.**fit(Xtrain,ytrain)  
yhat **=** SVM\_model**.**predict(Xtest)  
jacc\_acc\_score **=** jaccard\_score(ytest,yhat,average**=**'weighted')  
f1\_acc\_score **=** f1\_score(ytest,yhat,average**=**'weighted')  
  
print('Jaccard Score: ',jacc\_acc\_score)  
print('F1 Score: ',f1\_acc\_score)

Jaccard Score: 0.4508163265306122  
F1 Score: 0.5394383394383394

In [ ]:

result\_matriz**.**iloc[2,1] **=** jacc\_acc\_score  
result\_matriz**.**iloc[2,2] **=** f1\_acc\_score  
result\_matriz

Out[ ]:

|  | **Algorithm** | **Jaccard** | **F1 score** | **LogLoss** |
| --- | --- | --- | --- | --- |
| **0** | KNN | 0.572110 | 0.713618 | NA |
| **1** | Decision Tree | 0.489840 | 0.634498 | NA |
| **2** | SVM | 0.450816 | 0.539438 | NA |
| **3** | LogisticRegression | 0.000000 | 0.000000 | 0 |

Logistic Regression[¶](#3j2qqm3)

In [ ]:

**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.metrics **import** confusion\_matrix

In [ ]:

**%%capture**  
LR **=** LogisticRegression()  
param\_grid **=** {'penalty':['l1','l2','elasticnet'],'C':[0.1,1, 10, 100],  
 'solver':['newton-cg','lbfgs','liblinear','sag','saga']}  
LR **=** GridSearchCV(LR,param\_grid,refit**=True**,verbose**=**2)  
LR**.**fit(Xtrain,ytrain)

In [ ]:

print(LR**.**best\_estimator\_)

LogisticRegression(C=0.1, penalty='l1', solver='liblinear')

In [ ]:

*# Inicializamos los parametros obtenidos en el mejor estimador del modelo*  
C **=** 0.1  
penalty **=** 'l1'  
solver **=** 'liblinear'  
  
LR **=** LogisticRegression(C**=**C,penalty**=**penalty,solver**=**solver)  
LR**.**fit(Xtrain,ytrain)  
yhat **=** LR**.**predict(Xtest)  
  
*# Para el log loss tenemos que calcular las probabilidades de los valores predichos*  
yhat2 **=** LR**.**predict\_proba(Xtest)

In [ ]:

*# Calculamos las metricas para medir como performa el modelo*  
jacc\_acc\_score **=** jaccard\_score(ytest,yhat,average**=**'weighted')  
f1\_acc\_score **=** f1\_score(ytest,yhat,average**=**'weighted')  
log\_acc\_score **=** log\_loss(ytest,yhat2)  
  
print('Jaccard Score: ', jacc\_acc\_score)  
print('F1 Score: ', f1\_acc\_score)  
print('Log Loss: ', log\_acc\_score)

Jaccard Score: 0.4508163265306122  
F1 Score: 0.5394383394383394  
Log Loss: 0.5354259410746812

In [ ]:

result\_matriz**.**iloc[3,1] **=** jacc\_acc\_score  
result\_matriz**.**iloc[3,2] **=** f1\_acc\_score  
result\_matriz**.**iloc[3,3] **=** log\_acc\_score  
result\_matriz

Out[ ]:

|  | **Algorithm** | **Jaccard** | **F1 score** | **LogLoss** |
| --- | --- | --- | --- | --- |
| **0** | KNN | 0.572110 | 0.713618 | NA |
| **1** | Decision Tree | 0.489840 | 0.634498 | NA |
| **2** | SVM | 0.450816 | 0.539438 | NA |
| **3** | LogisticRegression | 0.450816 | 0.539438 | 0.535426 |

Model Evaluation using Test set[¶](#1y810tw)

In [ ]:

test\_df **=** pd**.**read\_csv('loan\_test.csv')  
test\_df**.**head()

Out[ ]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 50 | Bechalor | female |
| **1** | 5 | 5 | PAIDOFF | 300 | 7 | 9/9/2016 | 9/15/2016 | 35 | Master or Above | male |
| **2** | 21 | 21 | PAIDOFF | 1000 | 30 | 9/10/2016 | 10/9/2016 | 43 | High School or Below | female |
| **3** | 24 | 24 | PAIDOFF | 1000 | 30 | 9/10/2016 | 10/9/2016 | 26 | college | male |
| **4** | 35 | 35 | PAIDOFF | 800 | 15 | 9/11/2016 | 9/25/2016 | 29 | Bechalor | male |

First, download and load the test set:

In [ ]:

**!**wget -O loan\_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan\_test.csv

"wget" no se reconoce como un comando interno o externo,  
programa o archivo por lotes ejecutable.

Load Test set for evaluation[¶](#4i7ojhp)

In [ ]:

*# Read the dataset*  
test\_df **=** pd**.**read\_csv('loan\_test.csv')  
  
*# Transformations*  
test\_df['due\_date'] **=** pd**.**to\_datetime(df['due\_date'])  
test\_df['effective\_date'] **=** pd**.**to\_datetime(df['effective\_date'])  
test\_df['dayofweek'] **=** test\_df['effective\_date']**.**dt**.**dayofweek  
test\_df['weekend'] **=** test\_df['dayofweek']**.**apply(**lambda** x: 1 **if** (x**>**3) **else** 0)  
test\_df['Gender']**.**replace(to\_replace**=**['male','female'], value**=**[0,1],inplace**=True**)  
  
*# Select the main features*  
Feature\_test **=** test\_df[['Principal','terms','age','Gender','weekend']]  
Feature\_test **=** pd**.**concat([Feature\_test,pd**.**get\_dummies(test\_df['education'])], axis**=**1)  
Feature\_test**.**drop(['Master or Above'], axis **=** 1,inplace**=True**)  
Feature\_test**.**head()

Out[ ]:

|  | **Principal** | **terms** | **age** | **Gender** | **weekend** | **Bechalor** | **High School or Below** | **college** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1000 | 30 | 50 | 1 | 0 | 1 | 0 | 0 |
| **1** | 300 | 7 | 35 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1000 | 30 | 43 | 1 | 0 | 0 | 1 | 0 |
| **3** | 1000 | 30 | 26 | 0 | 1 | 0 | 0 | 1 |
| **4** | 800 | 15 | 29 | 0 | 1 | 1 | 0 | 0 |

In [ ]:

Feature**.**head()

Out[ ]:

|  | **Principal** | **terms** | **age** | **Gender** | **weekend** | **Bechalor** | **High School or Below** | **college** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1000 | 30 | 45 | 0 | 0 | 0 | 1 | 0 |
| **1** | 1000 | 30 | 33 | 1 | 0 | 1 | 0 | 0 |
| **2** | 1000 | 15 | 27 | 0 | 0 | 0 | 0 | 1 |
| **3** | 1000 | 30 | 28 | 1 | 1 | 0 | 0 | 1 |
| **4** | 1000 | 30 | 29 | 0 | 1 | 0 | 0 | 1 |

In [ ]:

yvalid **=** test\_df['loan\_status']**.**values  
*# xvalid = Feature\_test.values*  
Xvalid**=** preprocessing**.**StandardScaler()**.**fit(Feature\_test)**.**transform(Feature\_test)

In [ ]:

result\_validation **=** {'Algorithm':['KNN','Decision Tree','SVM','LogisticRegression'],  
'Jaccard':[0,0,0,0],'F1 score':[0,0,0,0],'LogLoss':['NA','NA','NA',0],}  
result\_validation **=** pd**.**DataFrame(result\_validation)  
result\_validation

Out[ ]:

|  | **Algorithm** | **Jaccard** | **F1 score** | **LogLoss** |
| --- | --- | --- | --- | --- |
| **0** | KNN | 0 | 0 | NA |
| **1** | Decision Tree | 0 | 0 | NA |
| **2** | SVM | 0 | 0 | NA |
| **3** | LogisticRegression | 0 | 0 | 0 |

KNN model - Validation[¶](#2xcytpi)

In [ ]:

knn **=** KNeighborsClassifier(n\_neighbors**=**6)  
knn**.**fit(Xtrain,ytrain)  
yresult **=** knn**.**predict(Xvalid)  
jacc\_knn\_valid **=** jaccard\_score(yvalid,yresult,average**=**'weighted')  
f1\_knn\_valid **=** f1\_score(yvalid,yresult,average**=**'weighted')  
print('Jaccard Score: ',jacc\_knn\_valid)  
print('F1 Score: ',f1\_knn\_valid)

Jaccard Score: 0.4876543209876543  
F1 Score: 0.6296296296296297

Decision Tree model - Validation[¶](#1ci93xb)

In [ ]:

decision\_tree **=** DecisionTreeClassifier(criterion **=** 'entropy', max\_depth**=**7)  
decision\_tree**.**fit(Xtrain,ytrain)  
yresult **=** decision\_tree**.**predict(Xvalid)  
jacc\_tree\_valid **=** jaccard\_score(yvalid,yresult,average**=**'weighted')  
f1\_tree\_valid **=** f1\_score(yvalid,yresult,average**=**'weighted')  
print('Jaccard Score: ',jacc\_tree\_valid)  
print('F1 Score: ',f1\_tree\_valid)

Jaccard Score: 0.5674786720115119  
F1 Score: 0.7097815764482431

SVM Model - Validation[¶](#3whwml4)

In [ ]:

yresult **=** SVM\_model**.**predict(Xvalid)  
jacc\_svm\_valid **=** jaccard\_score(yvalid,yresult,average**=**'weighted')  
f1\_svm\_valid **=** f1\_score(yvalid,yresult,average**=**'weighted')  
print('Jaccard Score: ',jacc\_svm\_valid)  
print('F1 Score: ',f1\_svm\_valid)

Jaccard Score: 0.5486968449931412  
F1 Score: 0.6304176516942475

Logistic Regression Model - Validation[¶](#2bn6wsx)

In [ ]:

yresult **=** LR**.**predict(Xvalid)  
yresult2 **=** LR**.**predict\_proba(Xvalid)  
jacc\_LR\_valid **=** jaccard\_score(yvalid,yresult,average**=**'weighted')  
f1\_LR\_valid **=** f1\_score(yvalid,yresult,average**=**'weighted')  
log\_loss\_LR\_valid **=** log\_loss(yvalid,yresult2)  
  
print('Jaccard Score: ',jacc\_LR\_valid)  
print('F1 Score: ',f1\_LR\_valid)  
print('Log Loss: ',log\_loss\_LR\_valid)

Jaccard Score: 0.5486968449931412  
F1 Score: 0.6304176516942475  
Log Loss: 0.5554920344372193

In [ ]:

result\_validation**.**iloc[0,1] **=** jacc\_knn\_valid  
result\_validation**.**iloc[0,2] **=** f1\_knn\_valid  
result\_validation**.**iloc[1,1] **=** jacc\_tree\_valid  
result\_validation**.**iloc[1,2] **=** f1\_tree\_valid  
result\_validation**.**iloc[2,1] **=** jacc\_svm\_valid  
result\_validation**.**iloc[2,2] **=** f1\_svm\_valid  
result\_validation**.**iloc[3,1] **=** jacc\_LR\_valid  
result\_validation**.**iloc[3,2] **=** f1\_LR\_valid  
result\_validation**.**iloc[3,3] **=** log\_loss\_LR\_valid  
result\_validation

Out[ ]:

|  | **Algorithm** | **Jaccard** | **F1 score** | **LogLoss** |
| --- | --- | --- | --- | --- |
| **0** | KNN | 0.487654 | 0.629630 | NA |
| **1** | Decision Tree | 0.567479 | 0.709782 | NA |
| **2** | SVM | 0.548697 | 0.630418 | NA |
| **3** | LogisticRegression | 0.548697 | 0.630418 | 0.555492 |

Conclusions[¶](#qsh70q)

### The best model for this problem is the decision tree, the best parameters are criterion = 'entropy' and max depth = 7[¶](#3as4poj)

### Thanks for reading[¶](#1pxezwc)

### Maximiliano Pona - Analytics Consultant[¶](#49x2ik5)

### LinkedIn: <https://www.linkedin.com/in/maximilianoezequielpona/>[¶](#2p2csry)

Report[¶](#147n2zr)

You should be able to report the accuracy of the built model using different evaluation metrics:

| **Algorithm** | **Jaccard** | **F1-score** | **LogLoss** |
| --- | --- | --- | --- |
| KNN | ? | ? | NA |
| Decision Tree | ? | ? | NA |
| SVM | ? | ? | NA |
| LogisticRegression | ? | ? | ? |

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: [SPSS Modeler](http://cocl.us/ML0101EN-SPSSModeler?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at [Watson Studio](https://cocl.us/ML0101EN_DSX?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

### Thanks for completing this lesson!

#### Author: [Saeed Aghabozorgi](https://ca.linkedin.com/in/saeedaghabozorgi?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01?utm_medium%3DExinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

[Saeed Aghabozorgi](https://ca.linkedin.com/in/saeedaghabozorgi), PhD is a Data Scientist in IBM with a track record of developing enterprise level applications that substantially increases clients’ ability to turn data into actionable knowledge. He is a researcher in data mining field and expert in developing advanced analytic methods like machine learning and statistical modelling on large datasets.

## Change Log[¶](#3o7alnk)

| **Date (YYYY-MM-DD)** | **Version** | **Changed By** | **Change Description** |
| --- | --- | --- | --- |
| 2020-10-27 | 2.1 | Lakshmi Holla | Made changes in import statement due to updates in version of sklearn library |
| 2020-08-27 | 2.0 | Malika Singla | Added lab to GitLab |

### © IBM Corporation 2020. All rights reserved.