Credit Card Approval

Aathithya-CB.EN.U4ELC20001
Adarsh-CB.EN.U4ELC20005
Arun-CB.EN.U4ELC20010

Summary of CA-1

- To classify people described by a set of attributes as good or bad credit risks for credit card applications.
- K-NN algorithm was used to achieve it.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- It can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

Summary of CA-2

- To classify people described by a set of attributes as good or bad credit risks for credit card applications.
- Used Principle Component Analysis(PCA) to compute covariance matrix, eigen vector and eigen values
- Covariance matrix was used to find the difference in dimensionality between the elements of the dataset
- Clustering was used on the elements of the dataset.
- Used normalization on the data and clustered elements using K-Means Clustering method.

References

- Credit Card Approval Dataset:

  https://www.kaggle.com/datasets/samuelcortinhas/credit-card-approval-clean-data

- Radial Basis Network:

  https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-are-radial-basis-functions-neural-networks

- SVM vs Neural Network:

  https://www.baeldung.com/cs/svm-vs-neural-network#:~:text=An%20SVM%20possesses%20a%20number,many%20layers%20as%20we%20want.

Problem Formulation

Objective: To classify people described by a set of attributes as good or bad credit risks for credit card applications using neural network.

Dataset details:

- No.of rows: 691
- No.of columns: 16
- No.of classes: 2
- Method of data collection is unknown

Link to full code mentioned in slides:

Assumptions:

- Feature- “Industry” was ignored while computing due to the presence of multiple strings and requires the help of NLP for solving.
- The feature “Ethnicity” was numeralised where:
  - White=0/Black=1/Asian=2/Latino=3/Other=4
- The feature “Gender” was numeralised where:
  - Female=0/Male=1
- The feature “Citizen” was numeralised where:
  - ByOtherMeans=0/ByBirth=1/Temporary=2

Neural Network application

- Visualizing the data

```
import numpy as np
import pandas as pd
df = pd.read_csv('/content/clean_dataset.csv')
df=df.drop(['Industry'], axis= 1)
df.head(15)
```

	Gender	Age	Debt	Married	BankCustomer	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore
0	1	30.83	0.000	1	1	0	1.250	1	1	1
1	0	58.67	4.460	1	1	1	3.040	1	1	6
2	0	24.50	0.500	1	1	1	1.500	1	0	0
3	1	27.83	1.540	1	1	0	3.750	1	1	5
4	1	20.17	5.625	1	1	0	1.710	1	0	0
5	1	32.08	4.000	1	1	0	2.500	1	0	0
6	1	33.17	1.040	1	1	1	6.500	1	0	0

```
#Display unique target/output values
print(np.unique(df['Approved']))
#Describing the dataset
df.describe()
```

```
[0 1]
```

	Gender	Age	Debt	Married	BankCustomer	Ethnicity	YearsEmployed	PriorDefault
count	690.000000	690.000000	690.000000	690.000000	690.000000	690.000000	690.000000	690.000000
mean	0.695652	31.514116	4.758725	0.760870	0.763768	0.781159	2.223406	0.523188
std	0.460464	11.860245	4.978163	0.426862	0.425074	1.151736	3.346513	0.499824
min	0.000000	13.750000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	22.670000	1.000000	1.000000	1.000000	0.000000	0.165000	0.000000
50%	1.000000	28.460000	2.750000	1.000000	1.000000	0.000000	1.000000	1.000000
75%	1.000000	37.707500	7.207500	1.000000	1.000000	1.000000	2.625000	1.000000
max	1.000000	80.250000	28.000000	1.000000	1.000000	4.000000	28.500000	1.000000

- Obtaining unique values for the target using mean, std, min of the given data.
- Count is equal to the number of rows = 690

```
[15] #Separating the features and output column in the dataset
     X = df.iloc[:,0:15]
     y = df.iloc[:,-1]
```

```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
#Selecting the best features from the dataset
kbest = SelectKBest(chi2,k=15)

best_features = kbest.fit(X,y)
best_features.scores_
```

```
[7] df_features = pd.DataFrame(best_features.scores_)
    df_columns = pd.DataFrame(X.columns)
```

- The features and output column (Approved) are separated
- Applying a score for each of the features based on their role in obtaining output.

```
[19] #Adding the selected features to the dataset
     featureScores = pd.concat([df_columns,df_features],axis=1)
```

```
#Display the features and their respective scores
featureScores.columns = ['Features','Score']
featureScores.sort_values(by='Score',ascending=False)
```

	Features	Score
13	Income	567228.629561
12	ZipCode	1145.725267
9	CreditScore	1121.332987
14	Approved	383.000000
6	YearsEmployed	360.893186
7	PriorDefault	170.746388
2	Debt	152.699990
8	Employed	82.965844
1	Age	82.802875
4	BankCustomer	5.820334
3	Married	5.380698
5	Ethnicity	3.910651
11	Citizen	0.879779

- The 15 features are arranged in descending order according to their respective scores.

```
[ ] X = df[['Income','ZipCode','CreditScore','Approved','YearsEmployed','PriorDefault','Debt','Employed','Age','BankCustomer',
    X
```

```
[ ]  #Display the dimensions of the features and output column
     X = X.values
     y = y.values
     print(X.shape,y.shape)
```

```
(690, 15)
```

- The dimensions of the dataset are calculated and printed.
- This is required to encode the values and plot them.

```
#Standardizing and normalising the values in the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
y = y.reshape(-1,1)
```

```
[ ] #Transforming the output column to hot encoded form
from sklearn.preprocessing import OneHotEncoder
ohot = OneHotEncoder()
y = ohot.fit_transform(y)
y = y.toarray()
```

```
[ ] #Splitting the dataset into testing and training parts
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

- The dataset values are standardized and normalized for preprocessing.
- Preprocessing is done by importing OneHotEncoder.
- The dataset is then split into testing(20%) and training parts.

```
[40] import keras
     from keras.models import Sequential
     from keras.layers import *
     import matplotlib.pyplot as plt
     %matplotlib inline
```

```
[41] #Training the network using Gaussian/Radial activation function
     model = Sequential()
     model.add(Dense(16,activation='gelu',input_dim = 15))
     model.add(Dense(8,activation='gelu'))
     model.add(Dense(2,activation='gelu'))
     model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 16)                256

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 2)                 18

=================================================================
Total params: 410
Trainable params: 410
Non-trainable params: 0
_________________________________________________________________
```

- Radial activation function is used and the trainable parameters are obtained.

```
[42] model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])

[55] #Displaying the loss accuracy, value loss and value accuracy for each epoch
     history = model.fit(X_train,y_train,epochs=5,validation_data=(X_test,y_test),batch_size=64)
```

```
Epoch 1/5
9/9 [==============================] - 0s 16ms/step - loss: 0.9244 - accuracy: 0.7826 - val_loss: 1.0449 - val_accuracy:
Epoch 2/5
9/9 [==============================] - 0s 16ms/step - loss: 0.8870 - accuracy: 0.7844 - val_loss: 1.0305 - val_accuracy:
Epoch 3/5
9/9 [==============================] - 0s 12ms/step - loss: 0.8969 - accuracy: 0.7862 - val_loss: 1.0394 - val_accuracy:
Epoch 4/5
9/9 [==============================] - 0s 18ms/step - loss: 0.9953 - accuracy: 0.7826 - val_loss: 1.0130 - val_accuracy:
Epoch 5/5
9/9 [==============================] - 0s 22ms/step - loss: 0.9578 - accuracy: 0.7862 - val_loss: 0.7619 - val_accuracy:
```

- The loss in accuracy, value and value accuracy is found for the dataset.
- The model is run for 5 epochs to calculate the accuracy here.

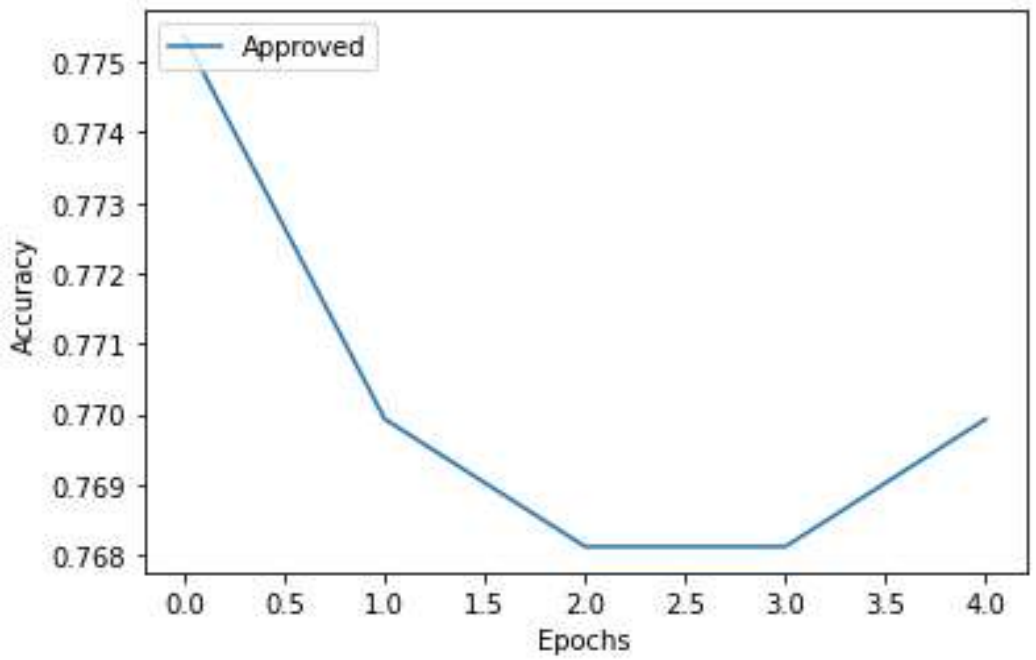
```
[56] y_pred = model.predict(X_test)

     #Inverse one hot encoding
     pred = []
     for i in range(len(y_pred)):
         pred.append(np.argmax(y_pred[i]))

     #Inverse encoding for y_test labels

     test = []
     for i in range(len(y_test)):
         test.append(np.argmax(y_test[i]))
```

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(pred,test)
print("Accuracy of Your Model is = " + str(acc*100))
```

```
Accuracy of Your Model is = 84.05797101449275
```

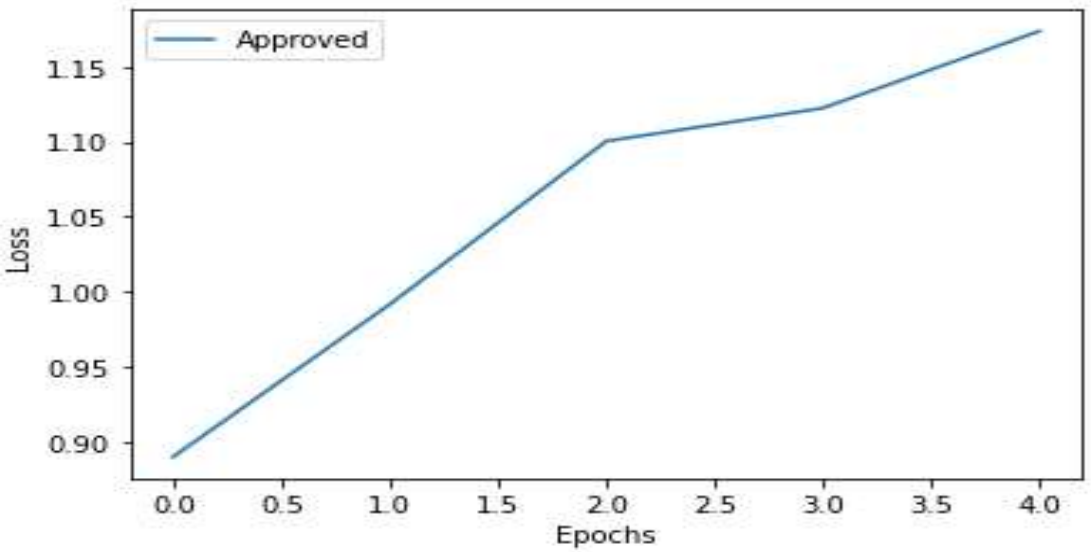
- The values are inverse encoded from one hot encoded(i.e., imported library) values.
- The accuracy value for the model is then displayed.

```
[64] plt.plot(history.history['accuracy'])
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.legend(['Approved'],loc='upper left')
     plt.show()
```



- The accuracy of the credit card approval is checked for the number of epochs done and plotted.

```
plt.plot(history.history['loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Features','Approved'],loc='upper left')
plt.show()
```



- The loss from credit card approval is checked for the number of epochs done and plotted.

Link for the Code:

https://colab.research.google.com/drive/17E0jbckTUy08hc3JlMYblmf708y6t4tG?usp=sharing

Link for the dataset:

https://www.kaggle.com/datasets/samuelcortinhas/credit-card-approval-clean-data

SVM

- Visualizing the data

```
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np

dataset = pd.read_csv('/content/clean_dataset (1).csv')
dataset.head(5)
```

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citizen	ZipCode	Income	Approved
0	1	30.83	0.000	1	1	Industrials	0	1.25	1	1	1	0	1	202	0	1
1	0	58.67	4.460	1	1	Materials	1	3.04	1	1	6	0	1	43	560	1
2	0	24.50	0.500	1	1	Materials	1	1.50	1	0	0	0	1	280	824	1
3	1	27.83	1.540	1	1	Industrials	0	3.75	1	1	5	1	1	100	3	1
4	1	20.17	5.625	1	1	Industrials	0	1.71	1	0	0	0	0	120	0	1

```
#Dropping the output column and the columns from the dataset which have binary values (0 or 1)
dset_modified = dataset.drop(['Married','Gender','PriorDefault','Employed','CreditScore','DriversLicense','Citizen','Approved'],axis=1) # dataset without class feature
data_set_feat = pd.DataFrame(dset_modified,columns=['Age','Debt','YearsEmployed','ZipCode','Income',])
data_set_feat
```

	Age	Debt	YearsEmployed	ZipCode	Income
0	30.83	0.000	1.25	202	0
1	58.67	4.460	3.04	43	560
2	24.50	0.500	1.50	280	824
3	27.83	1.540	3.75	100	3
4	20.17	5.625	1.71	120	0
...	...	...	...	...	...
685	21.08	10.085	1.25	260	0
686	22.67	0.750	2.00	200	394
687	25.25	13.500	2.00	200	1
688	17.92	0.205	0.04	280	750
689	35.00	3.375	8.29	0	0

- Columns which had binary values is dropped out and stored in modified dataset.
- Splitting of dataset for training and testing is done.

```
#Splitting the dataset into testing and training parts
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_set_feat,dataset['Married'], test_size = 0.20, random_state = 0)
```

```
from sklearn import svm
from sklearn.metrics import accuracy_score
model1 = svm.SVC(kernel='linear') #Linear Kernel
model2 = svm.SVC(kernel='rbf') #RBF Kernel
model3 = svm.SVC(kernel='poly') #Polynomial Kernel
model4 = svm.SVC(kernel='sigmoid') #Sigmoid Kernel
model5 = svm.SVC(gamma=10)
model6 = svm.SVC(gamma=10,C=1)

model1.fit(X_train,y_train)
model2.fit(X_train,y_train)
model3.fit(X_train,y_train)
model4.fit(X_train,y_train)
model5.fit(X_train,y_train)
model6.fit(X_train,y_train)

y_predModel1 = model1.predict(X_test)
y_predModel2 = model2.predict(X_test)
y_predModel3 = model3.predict(X_test)
y_predModel4 = model4.predict(X_test)
y_predModel5 = model5.predict(X_test)
y_predModel6 = model6.predict(X_test)

print("Accuracy of the Model 1: {0}%".format(accuracy_score(y_test, y_predModel1)*100))
print("Accuracy of the Model 2: {0}%".format(accuracy_score(y_test, y_predModel2)*100))
print("Accuracy of the Model 3: {0}%".format(accuracy_score(y_test, y_predModel3)*100))
print("Accuracy of the Model 4: {0}%".format(accuracy_score(y_test, y_predModel4)*100))
print("Accuracy of the Model 5: {0}%".format(accuracy_score(y_test, y_predModel5)*100))
print("Accuracy of the Model 6: {0}%".format(accuracy_score(y_test, y_predModel6)*100))
```

```
Accuracy of the Model 1: 80.43478260869566%
Accuracy of the Model 2: 80.43478260869566%
Accuracy of the Model 3: 80.43478260869566%
Accuracy of the Model 4: 81.88405797101449%
Accuracy of the Model 5: 80.43478260869566%
Accuracy of the Model 6: 80.43478260869566%
```

- Accuracy of the model is calculated

Inference

- Radial Basis Functions: They are a special class of feed-forward neural networks consisting of three layers: an input layer, a hidden layer, and the output layer.
- Support Vector Machine or SVM is a Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
- The main difference between SVM and NN is that SVM possesses a number of parameters that increase linearly with the linear increase in the size of the input. A Neural Network, on the other hand, does not.

- Neural network can be used for multiple layers unlike SVM.
- SVM can be of two of types:
  - Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line
  - Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line
- Non-linear SVM uses a kernel corresponds to a method for the reduction in the dimensionality of the input to a classifier.
- The kernel is, therefore, the function that's used in place of a dot product between two vectors
- The accuracy of the model changes as the number of epochs changes in Neural Networks.