Assignment for DAV Team -Head

Problem Statement 2.) Machine Learning

Code : https://github.com/Abhishek-mahajan02/DavH_Creditcard

Dataset

Default of credit card: This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005. Given *data about credit card clients*, let's try to predict whether a given client will **default** or not.

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:
X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).
X5: Age (year).
X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

Importing Libraries / Reading data

```
[3] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    %matplotlib inline
    import numpy as np
    import pandas as pd

    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split

    from sklearn.linear_model import LogisticRegression
    from sklearn.svm import SVC
```

```
[4] default = pd.read_csv('/content/default of credit card clients.csv')
```

```
default.head()
```

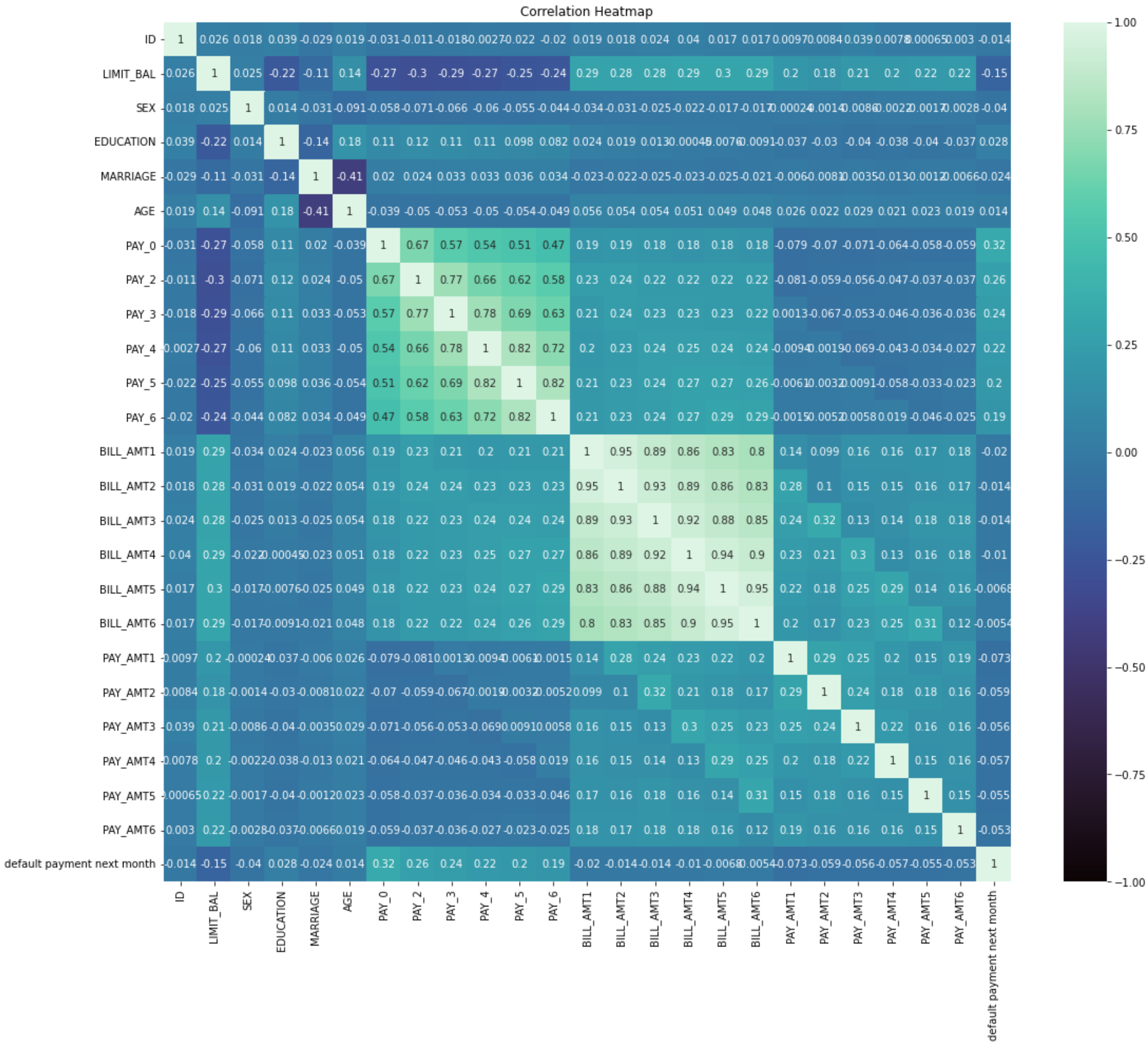
LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	de pa
20000	2	2	1	24	2	2	-1	-1	...	0	0	0	0	689	0	0	0	0	
120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	0	1000	1000	1000	0	2000	
90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	5000	
50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	2000	2019	1200	1100	1069	1000	
50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131	2000	36681	10000	9000	689	679	

‹ 25 columns

It is amazing data that easily tell whether a client may default or not in the next month's payment. we have another feature also. we only have to worry about scaling the data and little encoding as it is numerical data.

Visualization

```python
corr = default.corr()

plt.figure(figsize=(18, 15))
sns.heatmap(corr, annot=True, vmin=-1.0, cmap='mako')
plt.title("Correlation Heatmap")
plt.show()
```



Inference

We have 2 locations on the heatmap showing a lot of positive co-relations except age and marriage are negatively correlated. but the marital feature is a nominal value and a high value of marriage doesn't mean anything.

One imp. Non-Correlation id between column ["default payment next month "] with rest of the rows. This basically shows that if we are able to predict default payment next month, there is no dominance/contribution by any other feature individually but result of combination of all columns

Highly correlated between all the BILL_AMT1'S features basically tells that if one is default at BILL_AMT1 then it will most probably default at BILL_AMT2 and correlation increases as we move further till BILL_AMT6.

Data Preparation

X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).

These are nominal features and doesn't make sense to having different values . its not an order that 3>2 or 2>1 . so to make sure that every unique value should have its own column which can distinguish each feature, for that we are using 1 hot encoding.

```
default = pd.read_csv('/content/default of credit card clients.csv', index_col="ID")
default.rename(columns=lambda x: x.lower(), inplace=True)
# Base values: female, other_education, not_married
default['grad_school'] = (default['education'] == 1).astype('int')
default['university'] = (default['education'] == 2).astype('int')
default['high_school'] = (default['education'] == 3).astype('int')
default.drop('education', axis=1, inplace=True)

default['male'] = (default['sex']==1).astype('int')
default.drop('sex', axis=1, inplace=True)

default['married'] = (default['marriage'] == 1).astype('int')
default.drop('marriage', axis=1, inplace=True)

# For pay features if the <= 0 then it means it was not delayed
pay_features = ['pay_0','pay_2','pay_3','pay_4','pay_5','pay_6']
for p in pay_features:
    default.loc[default[p]<=0, p] = 0

default.rename(columns={'default payment next month':'default'}, inplace=True)
default
```

pay_amt3	pay_amt4	pay_amt5	pay_amt6	default	grad_school	university	high_school	male	married
0	0	0	0	1	0	1	0	0	1
1000	1000	0	2000	1	0	1	0	0	0
1000	1000	1000	5000	0	0	1	0	0	0
1200	1100	1069	1000	0	0	1	0	0	1

Building models using all features

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, precision_recall_curve
    from sklearn.preprocessing import RobustScaler
```

ROBUST SCALAR set all the variables to the same scale. which makes it an important pre-processing step.

Dropping the target column to get the training data (X), and the target column (y)

```
] target_name = 'default'
  X = default.drop('default', axis=1)
  robust_scaler = RobustScaler()
  X = robust_scaler.fit_transform(X)
  y = default[target_name]
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=123, stratify=y)
```

Train-Test Split: 85% of the entire dataset is used as the training, introducing a Random_state so that the same splits are created each time for reproducible results .

```
# Data frame for evaluation metrics
metrics = pd.DataFrame(index=['accuracy', 'precision', 'recall'],
                      columns=['NULL','LogisticReg', 'ClassTree', 'Naiv
eBayes']).
```

```
def CMatrix(CM, labels=['pay','default']):
    df = pd.DataFrame(data=CM, index=labels, columns=labels)
    df.index.name='TRUE'
    df.columns.name='PREDICTION'
    df.loc['Total'] = df.sum()
    df['Total'] = df.sum(axis=1)
    return df
```

```
Function to make good confusion matrix
```

- **Accuracy**: the proportion of the total number of predictions that are correct
- **Precision**: the proportion of positive predictions that are actually correct
- **Recall**: the proportion of positive observed values correctly predicted as such

In this application:

- **Accuracy**: Overall how often the model predicts correctly defaulters and non-defaulters
- **Precision**: When the model predicts **default**: how often is correct?
- **Recall**: The proportion of **actual defaulters** that the model will correctly predict as such

Which metric should I use?

- **False Positive**: A person who will pay predicted as a defaulter
- **False Negative**: A person who default predicted as payer

False negatives are worse => look for a better recall

Logistic Regression

```
# 1. Import the estimator object (model)
from sklearn.linear_model import LogisticRegression

# 2. Create an instance of the estimator
logistic_regression = LogisticRegression(n_jobs=-1, random_state=15)

# 3. Use the trainning data to train the estimator
logistic_regression.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = logistic_regression.predict(X_test)
metrics.loc['accuracy','LogisticReg'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','LogisticReg'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','LogisticReg'] = recall_score(y_pred=y_pred_test, y_true=y_test)
#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

PREDICTION	pay	default	Total
TRUE			
pay	3365	140	3505
default	671	324	995
Total	4036	464	4500

Confusion matrix for logistic regression

Classification tree

```
# 1. Import the estimator object (model)
from sklearn.tree import DecisionTreeClassifier

# 2. Create an instance of the estimator
class_tree = DecisionTreeClassifier(min_samples_split=30, min_samples_leaf=10, random_state=10)

# 3. Use the trainning data to train the estimator
class_tree.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = class_tree.predict(X_test)
metrics.loc['accuracy','ClassTree'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','ClassTree'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','ClassTree'] = recall_score(y_pred=y_pred_test, y_true=y_test)
#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

PREDICTION	pay	default	Total
TRUE			
pay	3185	320	3505
default	634	361	995
Total	3819	681	4500

Confusion Matrix for DesicionTreeClassifier with a minimum 30 number of observations that must be in a decision node to be able to split and control Overfitting.

Naïve Bayes Classifier

```
# 1. Import the estimator object (model)
from sklearn.naive_bayes import GaussianNB

# 2. Create an instance of the estimator
NBC = GaussianNB()

# 3. Use the trainning data to train the estimator
NBC.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = NBC.predict(X_test)
metrics.loc['accuracy','NaiveBayes'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','NaiveBayes'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','NaiveBayes'] = recall_score(y_pred=y_pred_test, y_true=y_test)

#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

PREDICTION	pay	default	Total
TRUE			
pay	2912	593	3505
default	439	556	995
Total	3351	1149	4500

Confusion Matrix for Naïve Base Classifier with GaussianNB because most of the features we work with are continuous features so this is the recommended model.

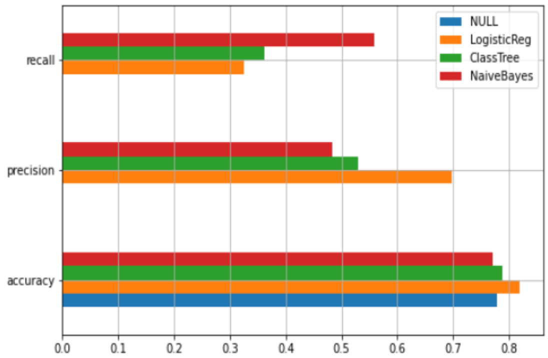
Metrics

```
[ ] 100*metrics
```

	NULL	LogisticReg	ClassTree	NaiveBayes
accuracy	77.888889	81.977778	78.8	77.066667
precision	0.0	69.827586	53.010279	48.389904
recall	0.0	32.562814	36.281407	55.879397

Visualization of these Metrics

```
fig, ax = plt.subplots(figsize=(8,5))
metrics.plot(kind='barh', ax=ax)
ax.grid();
```

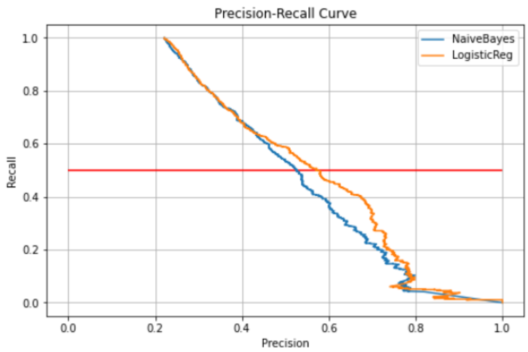


Till now it seems that Naïve Bayes is the best due to its higher recall and in terms of accuracy, all are nearly the same. buts are not final as we can Modify recalls by modifying the threshold.

Relationship between metrics for Naïve Bayes & Logistic Regression .

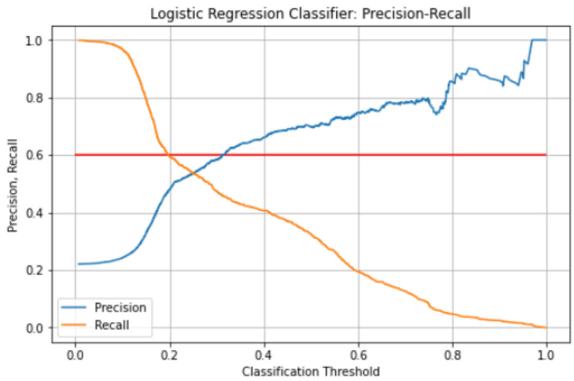
```
[ ] precision_nb, recall_nb, thresholds_nb = precision_recall_curve(y_true=y_test,
                                                                  probas_pred=NBC.predict_proba(X_test)[:,1])
    precision_lr, recall_lr, thresholds_lr = precision_recall_curve(y_true=y_test,
                                                                  probas_pred=logistic_regression.predict_proba(X_test)[:,1])
```

```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(precision_nb, recall_nb, label='NaiveBayes')
ax.plot(precision_lr, recall_lr, label='LogisticReg')
ax.set_xlabel('Precision')
ax.set_ylabel('Recall')
ax.set_title('Precision-Recall Curve')
ax.hlines(y=0.5, xmin=0, xmax=1, color='red')
ax.legend()
ax.grid();
```



From this we can see that little Logistic Regression is better because for a recall 0.5 Logistic Regression is having more precision than Naïve Bayes .

```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(thresholds_lr, precision_lr[1:], label='Precision')
ax.plot(thresholds_lr, recall_lr[1:], label='Recall')
ax.set_xlabel('Classification Threshold')
ax.set_ylabel('Precision, Recall')
ax.set_title('Logistic Regression Classifier: Precision-Recall')
ax.hlines(y=0.6, xmin=0, xmax=1, color='red')
ax.legend()
ax.grid();
```



Determining the variation of threshold with classification Precision , Recall .

Logistic Regression with threshold of 0.2 will be the best model

```
[ ] y_pred_proba = logistic_regression.predict_proba(X_test)[:,1]
    y_pred_test = (y_pred_proba >= 0.2).astype('int')
    #Confusion matrix
    CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
    print("Recall: ", 100*recall_score(y_pred=y_pred_test, y_true=y_test))
    print("Precision: ", 100*precision_score(y_pred=y_pred_test, y_true=y_test))
    CMatrix(CM)
```

```
Recall:  59.497487437185924
Precision:  47.85772029102668
```

PREDICTION	pay	default	Total
TRUE			
pay	2860	645	3505
default	403	592	995
Total	3263	1237	4500

Testing and making individual predictions

Logistic Regression with threshold 0.2 our fully trained model predict correctly that new costumer will default or Not default .

```
[ ] def make_ind_prediction(new_data):
        data = new_data.values.reshape(1, -1)
        data = robust_scaler.transform(data)
        prob = logistic_regression.predict_proba(data)[0][1]
        if prob >= 0.2:
            return 'Will default'
        else:
            return 'Will pay'
```

```
[ ] from collections import OrderedDict
    new_customer = OrderedDict([('limit_bal', 4000),('age', 50 ),('bill_amt1', 500),
                            ('bill_amt2', 35509 ),('bill_amt3', 689 ),('bill_amt4', 0 ),
                            ('bill_amt5', 0 ),('bill_amt6', 0 ), ('pay_amt1', 0 ),('pay_amt2', 35509 ),
                            ('pay_amt3', 0 ),('pay_amt4', 0 ),('pay_amt5', 0 ), ('pay_amt6', 0 ),
                            ('male', 1 ),('grad_school', 0 ),('university', 1 ), ('high_school', 0 ),
                            ('married', 1 ),('pay_0', -1 ),('pay_2', -1 ),('pay_3', -1 ),
                            ('pay_4', 0),('pay_5', -1), ('pay_6', 0)])

    new_customer = pd.Series(new_customer)
    make_ind_prediction(new_customer)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but RobustScaler was fitted with feature names
  "X does not have valid feature names, but"
'Will default'
```

Neural Networks

MLPClassifier

multilayer perceptron (MLP) is a feedforward artificial neural network model that maps input data sets to a set of appropriate outputs. An MLP consists of multiple layers and each layer is fully connected to the following one. The nodes of the layers are neurons with nonlinear activation functions, except for the nodes of the input layer. Between the input and the output layer there may be one or more nonlinear hidden layers.

```
[ ] models = {
        MLPClassifier():      "Neural Network"

    }

    for model in models.keys():
        model.fit(X_train, y_train)
    for model, name in models.items():
        print(name + ": {:.2f}%".format(model.score(X_test, y_test) * 100))

    Neural Network: 81.53%
```

Now given the simplicity of our dataset, we only use fully connected Deep Neural Networks for our task. Now, we identify the best architecture and method of training for the proposed network.

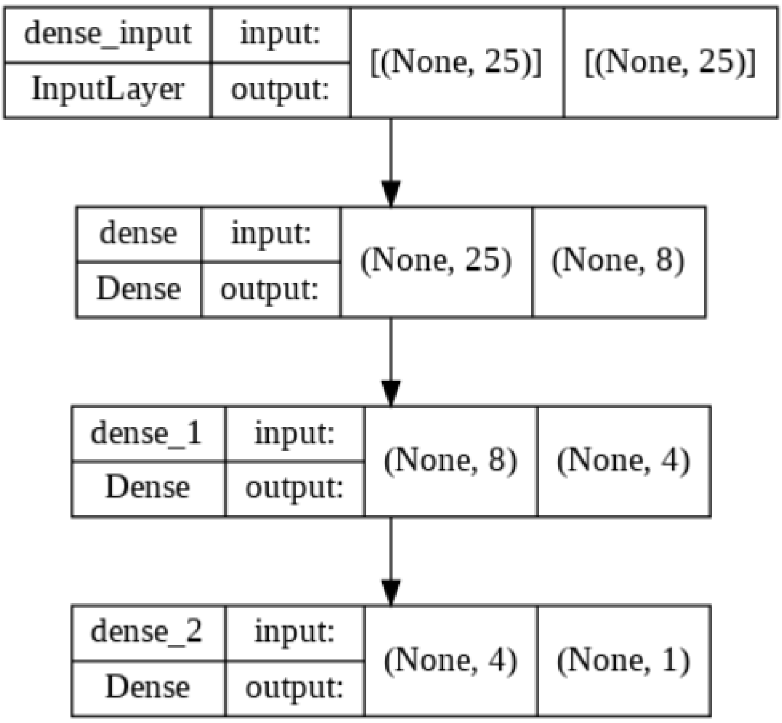
<u>Experiments on Architecture:</u>
We vary the model architecture, i.e., the number of hidden layers, and number of neurons within those layers, with the following hyperparameters:

I trained the data with two different Architectures .

MODEL 1

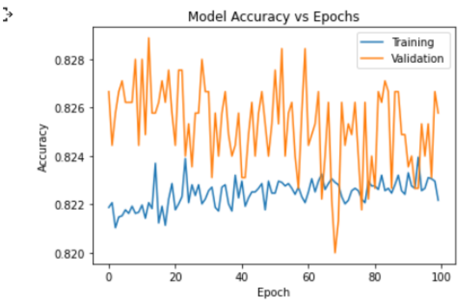
```
model = Sequential()
model.add(Dense(8, input_dim = len(X_train[0,:]) , activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 208

 dense_1 (Dense)             (None, 4)                 36

 dense_2 (Dense)             (None, 1)                 5

=================================================================
Total params: 249
Trainable params: 249
Non-trainable params: 0
_________________________________________________________________
None
```




```
Test loss: 0.43529120087623596
Test accuracy: 0.8167999982833862
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy vs Epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='best')

plt.show()
```



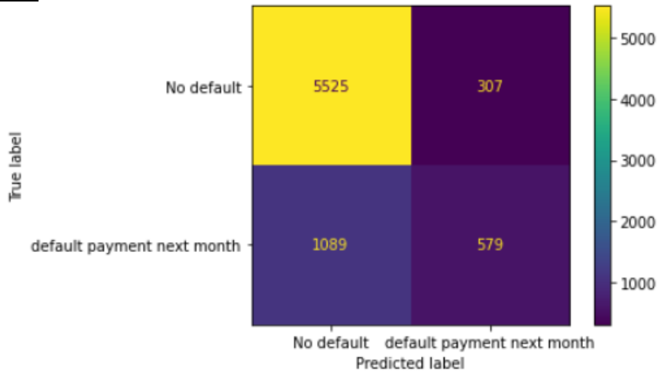
Batch size	Test F1 score	Test Accuracy
N=16	0.89	0.8162222504615784
N=32	0.89	0.8161333203315735
N=64	0.89	0.8172000050544739
N=128	0.89	0.8167999982833862

Test set performance on varying batch size (for learning rate = 0.001)

From these results, we select the **optimum batch size = 64 and learning rate = 0.001**.

```
model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])
history = model.fit(x=X_train, y=y_train, epochs=100, batch_size=64, validation_split = .1)

loss, accuracy = model.evaluate(x=X_test,y=y_test)

print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

Metrics

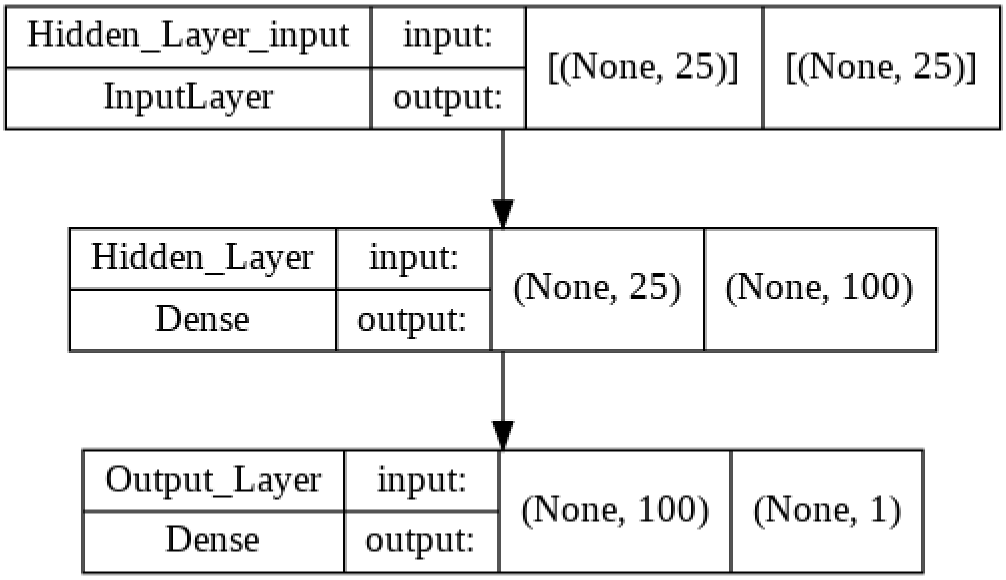



```
acuracy: 0.8172
precision: 0.6666666666666666
recall 0.35611510791366907
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      5832
           1       0.67      0.36      0.46      1668

    accuracy                           0.82      7500
   macro avg       0.75      0.65      0.68      7500
weighted avg       0.80      0.82      0.80      7500
```

MODEL 2

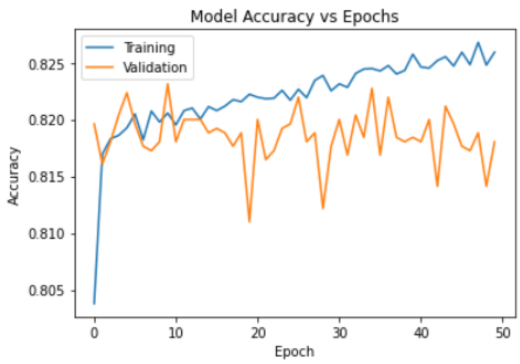
```
] model = Sequential(name="Neural_Network")
  model.add(Dense(100, activation = 'relu', name='Hidden_Layer'))
  model.add(Dense(1, activation = 'sigmoid',name='Output_Layer'))
  model.build((None,25))
```

```
Model: "Neural_Network"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Hidden_Layer (Dense)        (None, 100)               2600

 Output_Layer (Dense)        (None, 1)                 101

=================================================================
Total params: 2,701
Trainable params: 2,701
Non-trainable params: 0
_________________________________________________________________
```

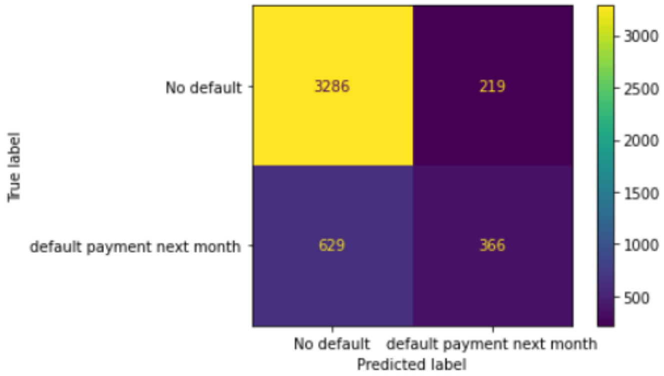



```
Test loss: 0.4375404417514801
Test accuracy: 0.8159999847412109
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy vs Epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='best')

plt.show()
```



```
Accuracy: 0.8115555555555556
Precision: 0.6256410256410256
Recall: 0.3678391959798995
F1-Score: 0.46329113924050636
```




```
acuracy: 0.8115555555555556
precision: 0.6256410256410256
recall 0.3678391959798995
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      3505
           1       0.63      0.37      0.46       995

    accuracy                           0.81      4500
   macro avg       0.73      0.65      0.67      4500
weighted avg       0.79      0.81      0.79      4500
```

From this we predict that Model 2 will be a better model as compared to Model 1 due to its less Varience and it seems a case of overfitting as there is too much fluctuations after every epoch during validation .