CSDA1010
Lab 2 Assignment

Classification models for the predictive
accuracy of probability of default of credit card clients

Group #2
Joshua Sheriff
Leah Jenkins
Vaibhav Chaudhary
Sarvesh Srivastava

Introduction/Business Understanding

Background

For credit card lending firms, mitigating the volume of customers who will default on their payments is of utmost importance. As more and more Canadians take on increasingly higher debt loads, the risks are rising regarding how many consumers will inevitably become delinquent in their monthly credit card payments. According to the Canadian Bankers Association, approx. 0.79% of Canadians had a delinquent credit card account (90 days or more overdue), resulting in an annualized 3.44% net loss rate in Q2 2019[1].

Looking across the way to Taiwan, credit card default rates spiked back in 2006[2], in large part due to credit being extended towards many unqualified individuals. For credit card issuers, it is a delicate balance to maintain between those who will dutifully pay monthly towards their bill and those who will run up charges and never intend to repay their debts. The ability to try and forecast or predict which customers may become delinquent early in the cycle would greatly give power back to the lenders and permit them to find new strategies to help recover losses and find ways to work with customers to prevent them from escalating further into debt collections.

Business Objective

The objective of our research is to devise three different classification models to help a credit card lender in Taiwan predict which customers will default on their payments in the future. By developing different classification models, the goal is to determine which model is more accurate in its prediction capabilities. In turn, this will permit the credit lender to make smarter decisions by being better able to evaluate which consumers have a better lifetime value to the credit company and also find new solutions on how to intervene earlier with customers who may eventually become delinquent payees.

Success Criteria

For this project, the success criteria is to determine which of 3 different classification methods (Naive Bayes, K-Nearest Neighbors and Support Vector Machines) will result in producing a statistically higher accuracy and prediction score.

Resources Inventory

- Project consultation team: Vaibhav, Sarvesh, Josh, Leah
- Data: Excel file containing 30,000 instances of credit card information containing 24 features Default of Credit Card Clients.
- Software resources: Jupyter Notebook (using Python language)
- Python packages: Pandas, Numpy, Matplotlib, Sci-kit Learn and Seaborn

[1] Canadian Bankers Association, Credit Card Delinquency - VISA and Mastercard, November 26, 2019, https://cba.ca/credit-card-delinquency-and-loss-statistics, Accessed December 9, 2019.
[2] Liang Yu et al., S&P Global, Will China's Credit Card Boom Follow the Well-worn Path to Bust?, July 4, 2019, https://www.spglobal.com/en/research-insights/articles/will-china-s-credit-card-boom-follow-the-well-worn-path-to-bust, Accessed December 10, 2019

Requirements, Assumptions & Constraints

For the purposes of our research, our team reviewed the datafile provided to ensure that there is no personal identifiable information (PII) that could provide insight into the customers from whom the data was originally sourced in Taiwan. The raw data table was made available for use courtesy of Yeh, I. C., & Lien, C. H. on the University of California Irvine's machine learning data repository.

For our use case, we assume that the data contained within the provided Excel file (see attachment) does not contain any clerical input errors (such as mistyped records) which could incorrectly bias or influence the dataset used in our learning algorithm.

During our data exploration phase (see further below), we realized that we were not entirely clear as to how it was determined whether a customer had defaulted or not defaulted on their debt. For our project, we are going to assume that the Y-variable (Default = 1, No Default = 0) is based on the actual outcome of that individual's real account history, and not based on a prior algorithm imputation or prediction.

Costs & Benefits

For credit lenders in Taiwan, the ability to better forecast and predict which customers may potentially default in the future is critical, as it would permit lenders to recoup more of the monies owed by customers who typically do not pay off enough of the balance each month, and help intervene earlier with customers who are trending towards default to find other ways to mitigate their debt load. The estimated loss of money due to default in Taiwan in February 2006 reached nearly $268 billion USD[3], and led not only to business repercussions, but societal as well - some of the individuals who became heavily indebted turned to drugs, became homeless or even tragically committed suicide[4].

As a credit account won't be declared as being in default until it reaches a period of 6 months in a row with missed minimum payments or no payments, by investing some upfront money into data models that can ingest large databases of data to help predict the likelihood of default, credit lenders would easily offset the cost of hiring data consultants to undertake this work.

Data Model Goals

Our team will be testing three different classification models - Naive Bayes, K-Nearest Neighbors and Support Vector Machines, to determine which of the three models will yield a higher probability of accuracy in determining whether someone will end up ultimately defaulting or not.

Model Success Criteria

For the Naive Bayes, KNN and SVM models, we are looking to achieve an accuracy score of at least 80% and a precision score of at least 75%.

[3] Eric Wang, Seven Pillars Institute, The Taiwan Credit Card Crisis, https://sevenpillarsinstitute.org/case-studies/taiwans-credit-card-crisis/, Accessed December 14, 2019

[4] Eric Wang, Seven Pillars Institute, The Taiwan Credit Card Crisis, https://sevenpillarsinstitute.org/case-studies/taiwans-credit-card-crisis/, Accessed December 14, 2019

Project Plan
For this project, we will be utilizing Python programming language and Jupyter notebook to help clean, evaluate and augment the data features and develop our models based on the dataset provided. Jupyter was selected as the software tool as it is easy to illustrate step-by-step the model development and share the results with the credit card company for future deployment.

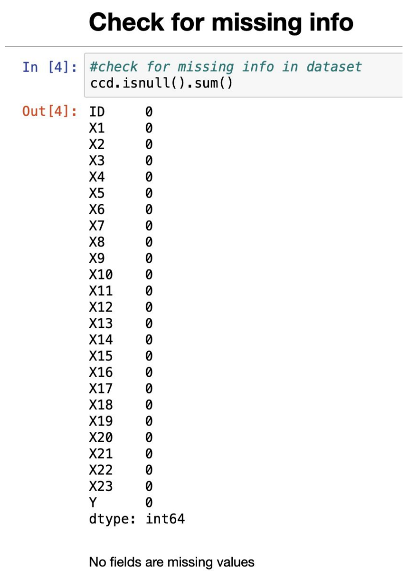
Data Understanding
Initial Collection Report
The credit card data was collected over a 6 month period in 2005 from 30,000 different customer records in Taiwan. Any details pertaining to PII is not contained within the file itself - only details such as gender, age, marital status and education level are within the dataset, but cannot be connected back to an individual person.

Data Description
As noted, the datafile is in Excel format. 24 different features have been included in the file:

Feature	**Interpretation**
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). 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.
X6	the repayment status in September, 2005.
X7	the repayment status in August, 2005.
X8	the repayment status in July, 2005.
X9	the repayment status in June, 2005.
X10	the repayment status in May, 2005.
X11	the repayment status in April, 2005.
X12-X17	Amount of bill statement (NT dollar).
X12	amount of bill statement in September, 2005
X13	amount of bill statement in August, 2005
X14	amount of bill statement in July, 2005
X15	amount of bill statement in June, 2005
X16	amount of bill statement in May, 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
X20	amount paid in July, 2005

X21	amount paid in June, 2005
X22	amount paid in May, 2005
X23	amount paid in April, 2005
Y	binary variable, default payment (Yes = 1, No = 0), as the response variable.

The full file contains 30,000 individual credit card data records. After initial review, we did not find any missing data fields in the file. The data has passed our requirements in order to use for the purposes of our research.



Check for missing info

```
In [4]: #check for missing info in dataset
        ccd.isnull().sum()

Out[4]: ID     0
        X1     0
        X2     0
        X3     0
        X4     0
        X5     0
        X6     0
        X7     0
        X8     0
        X9     0
        X10    0
        X11    0
        X12    0
        X13    0
        X14    0
        X15    0
        X16    0
        X17    0
        X18    0
        X19    0
        X20    0
        X21    0
        X22    0
        X23    0
        Y      0
        dtype: int64
```

No fields are missing values

Data Exploration

After importing the data, we did some basic descriptive statistics on the data file in order to understand the range of values, as shown in Figure 1.1. Looking at the values in X1 (credit limit), we saw that there was a wide range in the credit limit for customers, ranging from as little as NT$10,000 (approx. $435CAD) to as high as NT$1,000,000 (approx. $435,000CAD), with an average limit of NT$167,484. Features X2 through X5 represent the categorical features, such as gender (X2), education (X3), marital status (X4) and age (X5). The average age of a customer is 35 years old, with the eldest customer age being 79 and the youngest 21 years old.

The remaining features (X6-X23) represent the last 6 months (April-September) of repayment status, bill due amounts and amount paid. Across these features there is a range of values as seen in Figure 1.1.

Figure 1.1 Descriptive statistics on data features

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200	-0.220667
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868	1.169139
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	0.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	8.000000

X15	X16	X17	X18	X19	X20	X21	X22	X23	Y
30000.000000	30000.000000	30000.000000	30000.000000	3.000000e+04	30000.00000	30000.000000	30000.000000	30000.000000	30000.000000
43262.948967	40311.400967	38871.760400	5663.580500	5.921163e+03	5225.68150	4826.076867	4799.387633	5215.502567	0.221200
64332.856134	60797.155770	59554.107537	16563.280354	2.304087e+04	17606.96147	15666.159744	15278.305679	17777.465775	0.415062
-170000.000000	-81334.000000	-339603.000000	0.000000	0.000000e+00	0.00000	0.000000	0.000000	0.000000	0.000000
2326.750000	1763.000000	1256.000000	1000.000000	8.330000e+02	390.00000	296.000000	252.500000	117.750000	0.000000
19052.000000	18104.500000	17071.000000	2100.000000	2.009000e+03	1800.00000	1500.000000	1500.000000	1500.000000	0.000000
54506.000000	50190.500000	49198.250000	5006.000000	5.000000e+03	4505.00000	4013.250000	4031.500000	4000.000000	0.000000
891586.000000	927171.000000	961664.000000	873552.000000	1.684259e+06	896040.00000	621000.000000	426529.000000	528666.000000	1.000000

We then decided to do some further exploration of gender, education, marital status and age features to help us understand the significance of this data.

Marital Status: Looking at the marital status feature further, we discovered that there were 4 categories: 1 = married, 2 = single, 3 = other and a 4th variable (value of 0), which had not been correctly identified in the original file. We decided to categorize the 54 unknown values as "other" and group them with category 3, as it is unclear as to what the true status of these individuals are. This is not expected to result in any skew of the data overall.

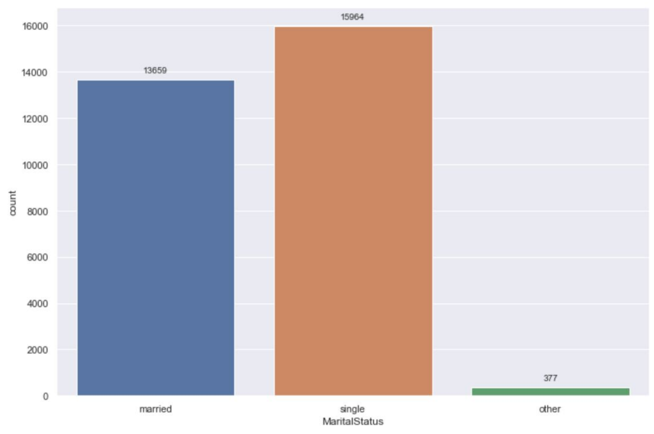
```
ccd['X4'].unique()
```

```
array([1, 2, 3, 0], dtype=int64)
```

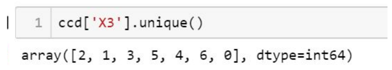
```
demographics.groupby('MaritalStatus').count()
```

MaritalStatus	CreditLimit	Gender	Education	Age	NextPayment
married	13659	13659	13659	13659	13659
other	377	377	377	377	377
single	15964	15964	15964	15964	15964

Figure 1.2 Count by Marital Status Categories



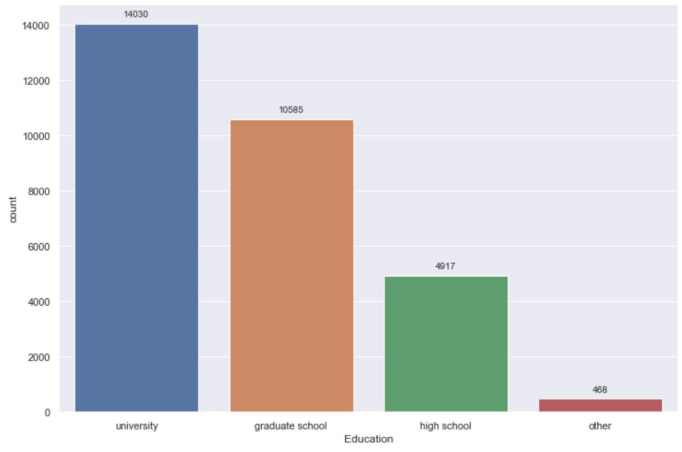
Education: Exploring the education features, similar to the marital status, we noted that there appeared to be 3 additional categories (0, 5 and 6) present in the data that were not correctly identified/labeled in the original data. The education categories are: 0 = unknown, 1 = graduate school, 2 = university, 3 = high school, 4 = others, 5 = unknown, 6 = unknown. We opted to group the 345 unknown entries and add them to category 4 of “others”, which now represents around 1.6% of all values. Similar to before, we do not believe that re-categorizing and grouping the unknown features will result in any significant skew of the data.



```
ccd['X3'].unique()
```

```
array([2, 1, 3, 5, 4, 6, 0], dtype=int64)
```

Figure 1.3 Count by Education Category



Gender: Looking at the gender variable, we saw that the data was split 60% female and 40% male (Figure 1.4). When we further split the gender across those who ultimately defaulted vs did not default (Figure 1.5), we noted that males had a higher likelihood to default (24%) compared to females (20%).

Figure 1.4 Comparison of Female vs Male customers

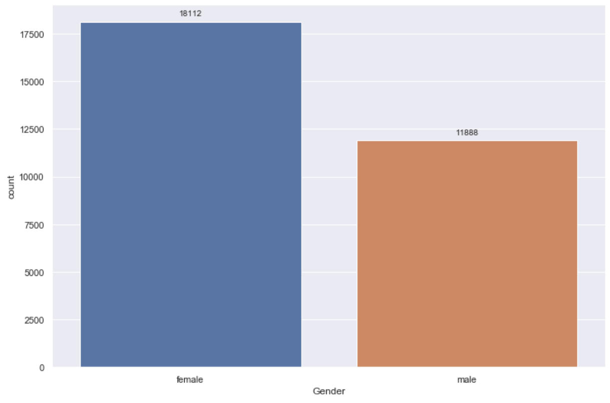
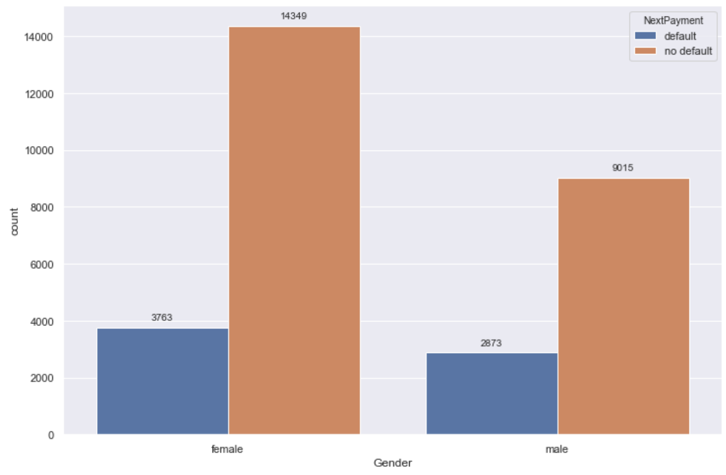
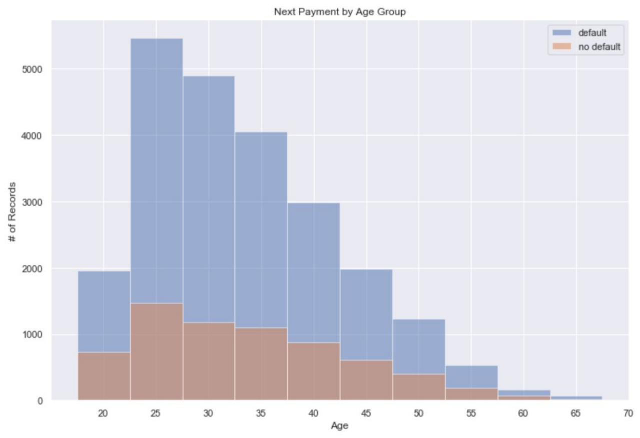


Figure 1.5 Default rate by gender



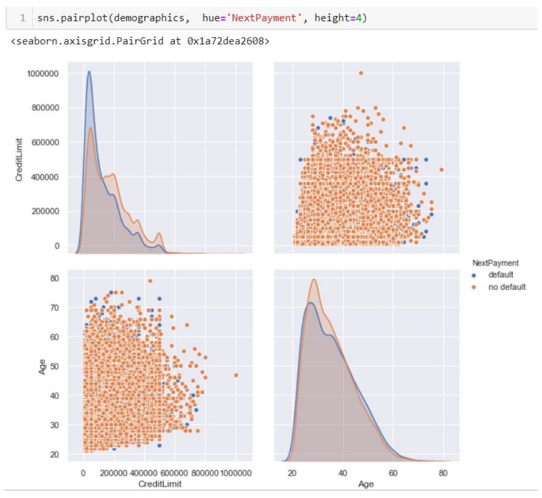
When plotting the credit card holders by age, we discovered that the largest group to default on their next payment is the 25-29 year old age group (Figure 1.6), which may be a reflection of younger people not having as much experience in handling their finances and debt compared to an older demographic. Looking at the 50-54 age group, approximately 75% of them will not have their account go into default on their upcoming bill, but when you move out to the 60-64 age group, that drops down to only 69% and goes back up to 75% among the 65-69 group.

Figure 1.6 Next payment status by age group



To understand how customers are defaulting across different features, we created a pairplot to help understand if there are large variances between the two groups of customers. Looking at Figure 1.7, we saw that the age of defaulters vs non-defaulters are very similar to one another, while those with higher credit limits skew more towards not defaulting on their next payment.

Figure 1.7 Pairplot of credit limit and age by next payment status

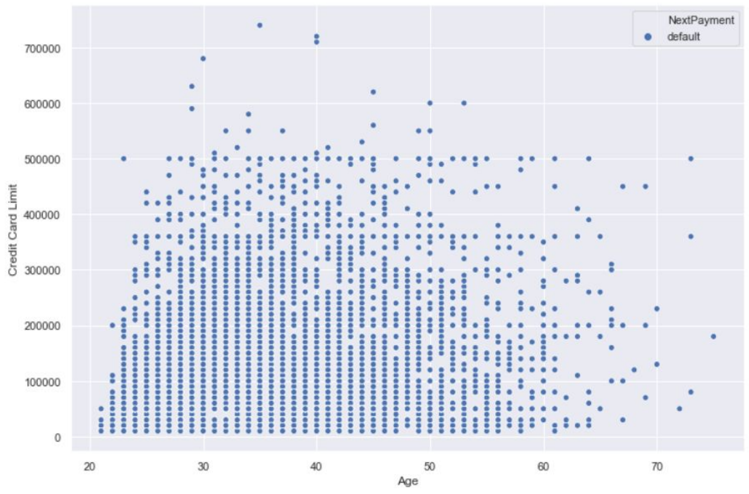


```
sns.pairplot(demographics,  hue='NextPayment', height=4)
```

```
<seaborn.axisgrid.PairGrid at 0x1a72dea2608>
```

Further exploring those who ultimately defaulted, we created a scatter plot (Figure 1.8) of their age vs credit card limit to understand and identify any outliers in the default data. There is a high concentration of defaulting among those with credit limits up to NT$400,000, but as credit limits increase, the likelihood of defaulting decreases. This could perhaps be an indication that some of these credit accounts might be associated with business use rather than individual consumption or perhaps that those with really high limits seldom use up all the available credit room and are less likely to end up defaulting on repayments.

Figure 1.8 Age vs credit limit amongst account defaulters



Data Preparation

Construct new features

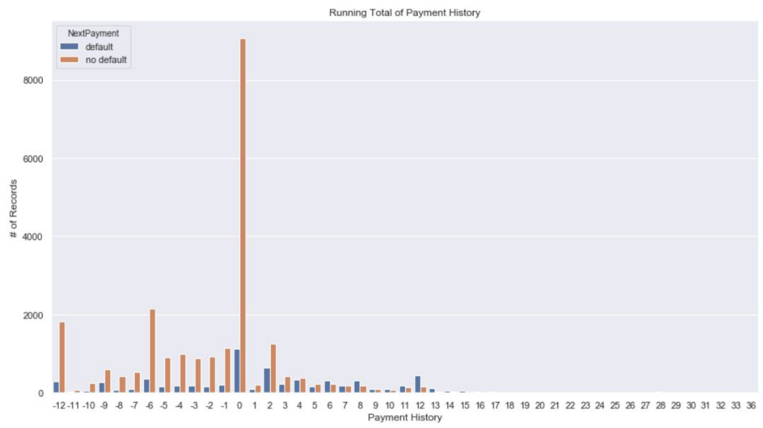
Prior to determining which features to include in our three models, we looked to augment our dataset by creating some new features that would help us condense the bill payment history, as well as create new ratio features that could help us understand credit limit utilization and bill payment utilization.

Bill Payment History: The first variable we devised was to create a running total of payment history to determine how many months in arrears or on time a customer has been in paying their monthly credit bill. Using the running total of months helps us identify against the dataset how many months in arrears customers who ultimately defaulted were falling behind. The negative numbers shown in Figure 1.9 reflect that the customer was making payments, while positive numbers indicate how many months indebted customers had become.

2.4.1 Create a running total of payment history and plot against defaulted or not

```
ccd['paymentHist'] = ccd['X6'] + ccd['X7'] + ccd['X8'] + ccd['X9'] + ccd['X10'] + ccd['X11']
financials=ccd[['X1', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11','paymentHist', 'Y']].copy()
financials.rename(columns={"Y":"NextPayment"}, inplace=True)
financials['NextPayment'] = np.where(financials['NextPayment']==1, 'default', 'no default')
```

Figure 1.9 Running Total of Payment History variable

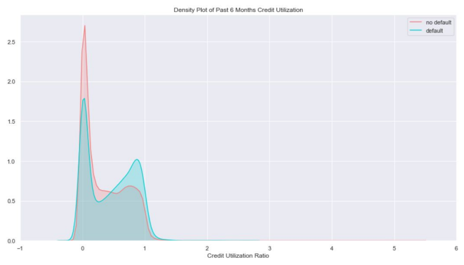


Credit Utilization Ratio: The next variable that we created was a credit utilization ratio, meant to help us understand how much of their total available credit customers were using over the 6 month period. As illustrated in Figure 1.10, we built a density plot of this new credit utilization ratio and found that among customers who did not default, they tended to have a higher credit utilization ratio comparative to those who defaulted.

2.4.2 Create ratio of previous 6 month statement balance over credit limit (Credit Utilization)

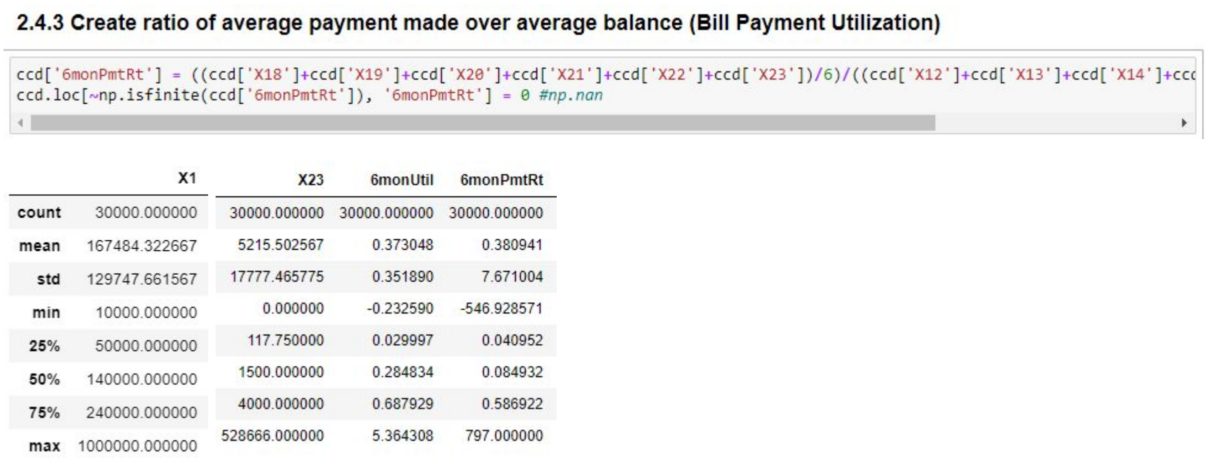
```
ccd['6monUtil'] = ((ccd['X12']+ccd['X13']+ccd['X14']+ccd['X15']+ccd['X16']+ccd['X17'])/6)/ccd['X1']
```

Figure 1.10 Credit card utilization ratio



6 Month Payment Ratio: Lastly, we calculated a ratio to understand the average payment made over the average balance (across the 6 month timeframe) to help us understand how people are typically paying their bills - aka bill payment utilization. Reviewing the stats on this new metric (Figure 1.11), we saw that on average customers were paying back 38% of their balance, with those in the first quartile only paying back 4.1% and those in the third quartile were repaying closer to 58.7% of their balance.

Figure 1.11 Statistics on bill payment utilization ratio (6monPmRt)



2.4.3 Create ratio of average payment made over average balance (Bill Payment Utilization)

```
ccd['6monPmtRt'] = ((ccd['X18']+ccd['X19']+ccd['X20']+ccd['X21']+ccd['X22']+ccd['X23'])/6)/((ccd['X12']+ccd['X13']+ccd['X14']+ccc
ccd.loc[~np.isfinite(ccd['6monPmtRt']), '6monPmtRt'] = 0 #np.nan
```

	X1	X23	6monUtil	6monPmtRt
count	30000.000000	30000.000000	30000.000000	30000.000000
mean	167484.322667	5215.502567	0.373048	0.380941
std	129747.661567	17777.465775	0.351890	7.671004
min	10000.000000	0.000000	-0.232590	-546.928571
25%	50000.000000	117.750000	0.029997	0.040952
50%	140000.000000	1500.000000	0.284834	0.084932
75%	240000.000000	4000.000000	0.687929	0.586922
max	1000000.000000	528666.000000	5.364308	797.000000

Data Selection Rationale

To help us determine which mix of X-features would be better suited for each of the three models, we decided to run recursive feature elimination (RFE). The goal of recursive feature elimination is to determine which features are the highest performer (based on their coefficients) by running multiple model iterations to exhaust each feature - it is sometimes referred to as greedy optimization.

We ran 2 iterations of RFE - the first using our dataset where we leveraged the “Get Dummies” function to convert our categorical features (for use with our SVM and KNN models) and a second version converting our categorical features into integers (for use with Naive Bayes). Interestingly, we noticed that it suggested using only the Gender_Female variable rather than both gender features. We believe that this may be due to using the “Get Dummies” function splitting the genders into 2 distinct features on their own that the RFE determined that it really didn’t require using both since the use of a 0 or 1 within the Gender_Female variable would already indicate that they were or weren’t a woman (essentially just the inverse of one another).

We then created correlation matrices for both versions of our dataset (used in the RFE) to understand which of our select features had the most collinearity. In Figure 1.12, we see that there is more collinearity between the repayment status in September (X6), repayment status in June (X9), repayment status in April (X11) and the running payment history. This would seem to suggest that the repayment status of every couple of months seems to have more impact on their future payment status.

The second correlation matrix (Figure 1.13) which was based on the dataset used for our Naive Bayes model, also showed a lot of correlation between the repayment statuses in different months (X6-X11), while there was little to no interdependency between our categorical features such as gender, education or marital status.

4.1.1.1 Using dataset (ccd3) where get dummies function was used to convert categorical into dummy/indicator variables

```python
#using dataset (ccd3) where get dummies function was used to convert categorical into dummy/indicator variables
rfe_X_cols = ['ID', 'CreditLimit', 'Age', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12',
        'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20', 'X21', 'X22',
        'X23', 'paymentHist', '6monUtil','6monPmtRt',
        'Gender_female', 'Gender_male', 'Education_graduate school',
        'Education_high school', 'Education_other', 'Education_university',
        'MaritalStatus_married', 'MaritalStatus_other', 'MaritalStatus_single']
rfe_X = ccd3[rfe_X_cols]
rfe_y = ccd3['NextPayment']
ccd3
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.feature_selection import RFE

# Build a logreg and compute the feature importances
model = LogisticRegression(solver='liblinear', max_iter=500)
# create the RFE model and select 8 attributes
rfe = RFE(model,10)
rfe = rfe.fit(rfe_X, rfe_y)
# summarize the selection of the attributes
print('Selected features: %s' % list(rfe_X.columns[rfe.support_]))
```

```
Selected features: ['X6', 'X9', 'X11', 'paymentHist', '6monUtil', 'Gender_female', 'Education_graduate school', 'Education_h
igh school', 'Education_university', 'MaritalStatus_single']
```

```python
rfe_X_sug_cols = ['X6', 'X9', 'X11', 'paymentHist', '6monUtil', 'Gender_female', 'Education_graduate school', 'Education
rfe_X_sug_cols
```

```
['X6',
 'X9',
 'X11',
 'paymentHist',
 '6monUtil',
 'Gender_female',
 'Education_graduate school',
 'Education_high school',
 'Education_university',
 'MaritalStatus_single']
```

4.1.1.2 Using dataset (ccd4) where categorical values were converted to integers

```python
#using dataset (ccd4) where categorical values were converted to integers.
rfe_X_gnb_cols = ['ID', 'CreditLimit', 'Gender', 'Education', 'MaritalStatus', 'Age',
        'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16',
        'X17', 'X18', 'X19', 'X20', 'X21', 'X22', 'X23', 'paymentHist', '6monUtil', '6monPmtRt']
rfe_X_gnb = ccd4[rfe_X_gnb_cols]
rfe_y_gnb = ccd4['NextPayment']
```

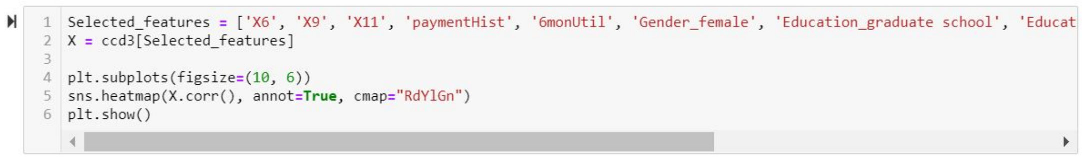
```python
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.feature_selection import RFE

# Build a logreg and compute the feature importances
model = LogisticRegression(solver='liblinear', max_iter=500)
# create the RFE model and select 8 attributes
rfe = RFE(model,10)
rfe = rfe.fit(rfe_X_gnb, rfe_y_gnb)
# summarize the selection of the attributes
print('Selected features: %s' % list(rfe_X_gnb.columns[rfe.support_]))
```

```
Selected features: ['Gender', 'Education', 'MaritalStatus', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'paymentHist']
```

```python
#setup test/train/split dataframe for GNB using suggested features
rfe_X_gnb_sug_cols = ['Gender', 'Education', 'MaritalStatus', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'paymentHist']
rfe_X_gnb_sug_cols
```

```
1]: ['Gender',
     'Education',
     'MaritalStatus',
     'X6',
     'X7',
     'X8',
     'X9',
     'X10',
     'X11',
     'paymentHist']
```

Figure 1.12 Correlation matrix (SVM & KNN)



```
Selected_features = ['X6', 'X9', 'X11', 'paymentHist', '6monUtil', 'Gender_female', 'Education_graduate school', 'Educat
X = ccd3[Selected_features]

plt.subplots(figsize=(10, 6))
sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
plt.show()
```

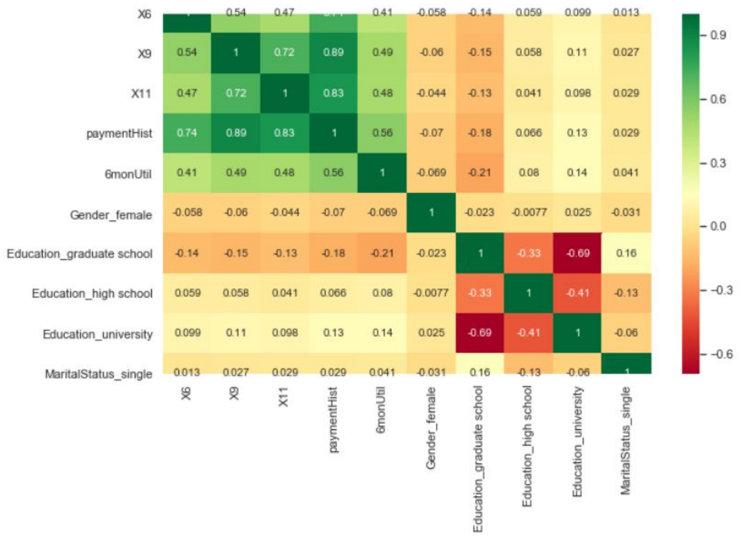
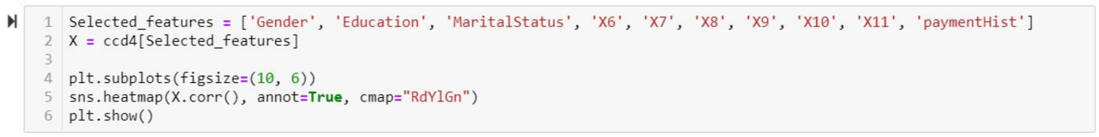
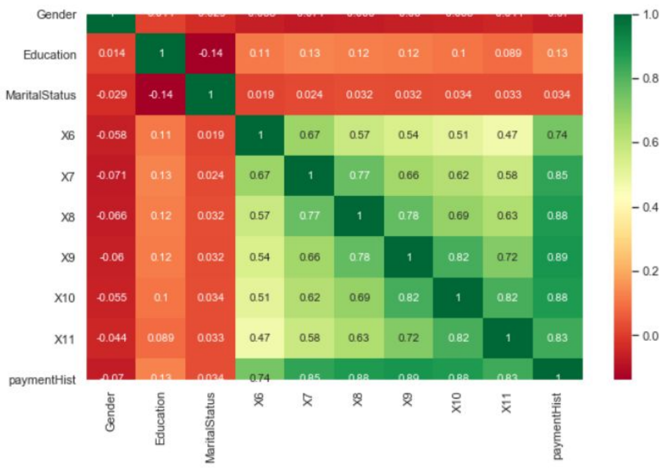


Figure 1.13 Correlation matrix (Naive Bayes)

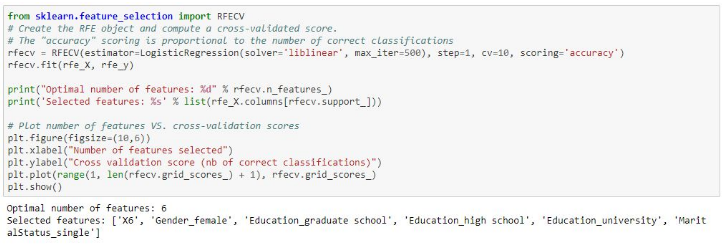


```
Selected_features = ['Gender', 'Education', 'MaritalStatus', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'paymentHist']
X = ccd4[Selected_features]

plt.subplots(figsize=(10, 6))
sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
plt.show()
```



Figures 1.14 and 1.15 illustrate both RFEs using cross-validation to help evaluate the number of recommended features for our models.

Figure 1.14 RFE and cross-validation (SVM & KNN)



```
from sklearn.feature_selection import RFECV
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct classifications
rfecv = RFECV(estimator=LogisticRegression(solver='liblinear', max_iter=500), step=1, cv=10, scoring='accuracy')
rfecv.fit(rfe_X, rfe_y)

print("Optimal number of features: %d" % rfecv.n_features_)
print('Selected features: %s' % list(rfe_X.columns[rfecv.support_]))

# Plot number of features VS. cross-validation scores
plt.figure(figsize=(10,6))
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

```
Optimal number of features: 6
Selected features: ['X6', 'Gender_female', 'Education_graduate school', 'Education_high school', 'Education_university', 'Marit
alStatus_single']
```

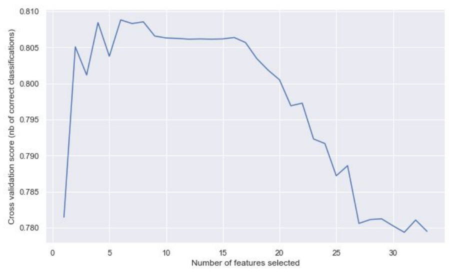
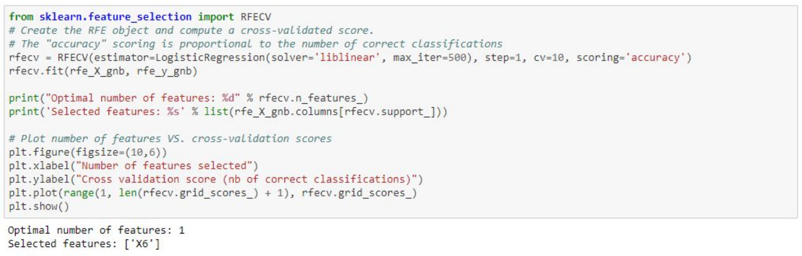
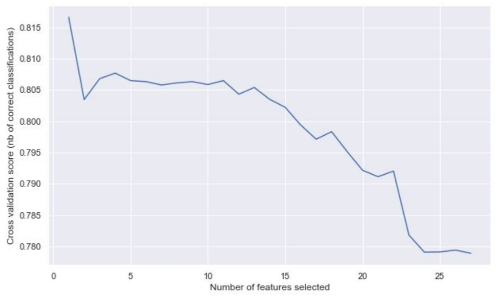


Figure 1.15 RFE and cross-validation (Naive Bayes)



```
from sklearn.feature_selection import RFECV
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct classifications
rfecv = RFECV(estimator=LogisticRegression(solver='liblinear', max_iter=500), step=1, cv=10, scoring='accuracy')
rfecv.fit(rfe_X_gnb, rfe_y_gnb)

print("Optimal number of features: %d" % rfecv.n_features_)
print('Selected features: %s' % list(rfe_X_gnb.columns[rfecv.support_]))

# Plot number of features VS. cross-validation scores
plt.figure(figsize=(10,6))
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

```
Optimal number of features: 1
Selected features: ['X6']
```



Feature Selection

For the Naive Bayes algorithm, we will be using the following X-features:

- Gender, Education, MaritalStatus, X6, X7, X8, X9, X10, X11, paymentHist

For the SVM and KNN algorithms, we will be using the following X-features:

- X6, Gender_female, Education_graduate school, Education_high school, Education_university, MaritalStatus_single

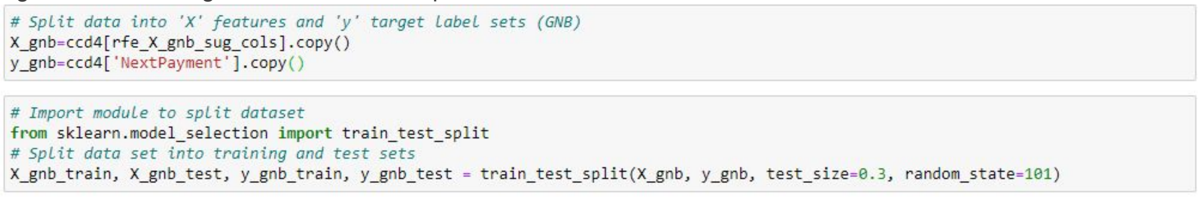
All three models will be using the same Y-feature of Default (1) or No Default (0).

Modeling Phase

Based on reviewing all of the features in our dataset, since we already have the binary output variable of whether someone defaulted on their next payment (0) or did not default (1), our three data models will be based on classification using supervised learning. As noted earlier, we determined that for our classification algorithms we would test three different models - Naive Bayes, K-Nearest Neighbors and Support Vector Machines to determine which of these would get us to an accuracy prediction of 80% or higher with a precision of at least 75%.

Naive Bayes: The Naive Bayes classifier works off the Bayes theorem, working on the assumption of independence between the different predictor features.[5] It is a rather quick classifier and was the first option we sought to use to help predict whether a credit user would end up defaulting or not.

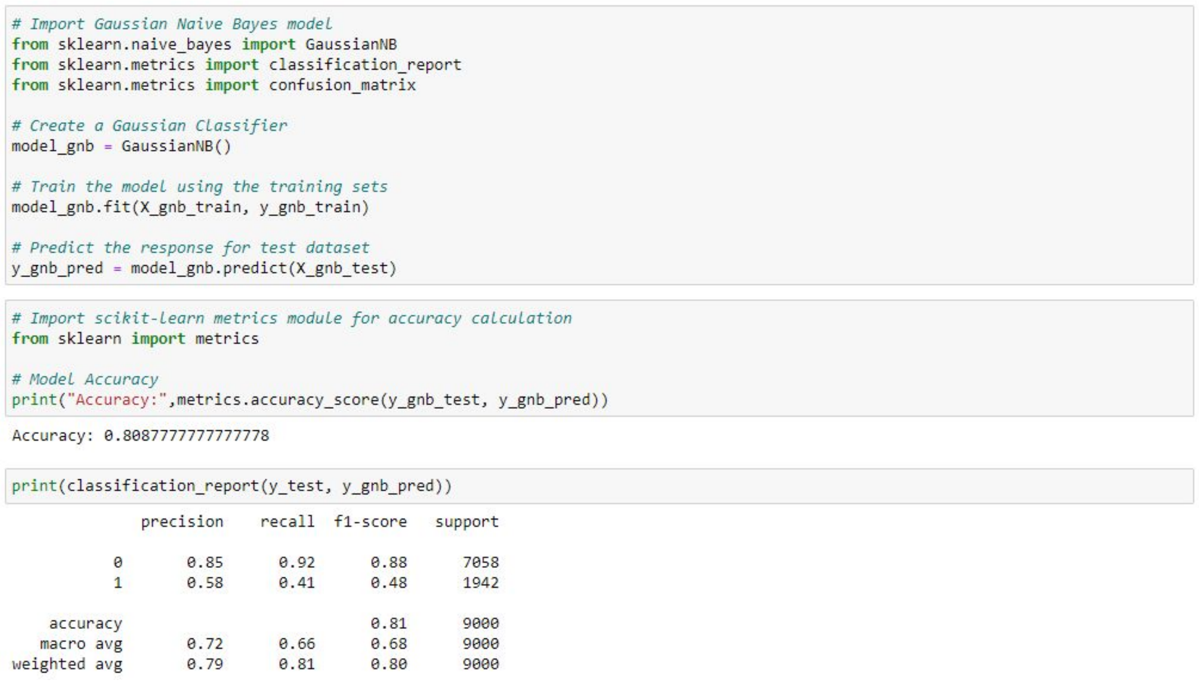
Figure 1.16 Creating test and train data split



```
# Split data into 'X' features and 'y' target label sets (GNB)
X_gnb=ccd4[rfe_X_gnb_sug_cols].copy()
y_gnb=ccd4['NextPayment'].copy()
```

```
# Import module to split dataset
from sklearn.model_selection import train_test_split
# Split data set into training and test sets
X_gnb_train, X_gnb_test, y_gnb_train, y_gnb_test = train_test_split(X_gnb, y_gnb, test_size=0.3, random_state=101)
```

Figure 1.17 Building the model



```
# Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Create a Gaussian Classifier
model_gnb = GaussianNB()

# Train the model using the training sets
model_gnb.fit(X_gnb_train, y_gnb_train)

# Predict the response for test dataset
y_gnb_pred = model_gnb.predict(X_gnb_test)
```

```
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_gnb_test, y_gnb_pred))
```

```
Accuracy: 0.8087777777777778
```

```
print(classification_report(y_test, y_gnb_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      7058
           1       0.58      0.41      0.48      1942

    accuracy                           0.81      9000
   macro avg       0.72      0.66      0.68      9000
weighted avg       0.79      0.81      0.80      9000
```

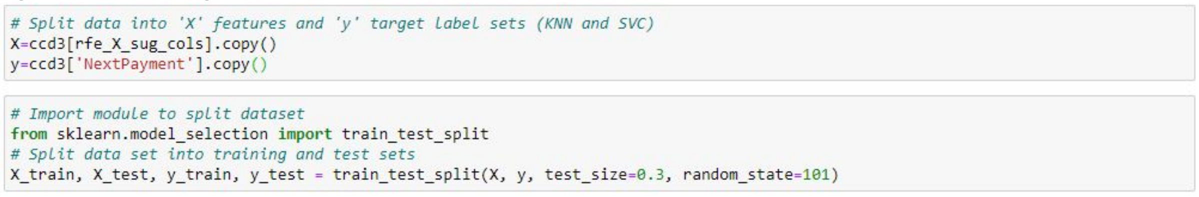
Looking at the output, we saw that the Naive Bayes model produced a high degree accuracy, achieving 81%. When we look at the precision of this model, it showed 58% as its ratio of correctly predicting default (“positive”) compared to all predicted positives, which would suggest that the model is a bit above average. The recall ratio for default is on the lower side (41%), and overall the F1-Score achieved was 48%. As a simple approach, Naive Bayes does offer a fairly accurate method of prediction and is known to be a very cost

[5] Sunil Ray, Analytics Vidhya, 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R, September 11, 2017 https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/, Accessed December 18, 2019

effective model to employ against large datasets. On the downside, it does assume that all the features are independent from one another, which in the real world often likely isn't the case.

KNN: K-Nearest Neighbors algorithm is based on the premise that like things tend to be clustered around one another. It functions by inputting a K-factor to determine what number of closest neighbours to a new value point it should use to help it classify.[6]

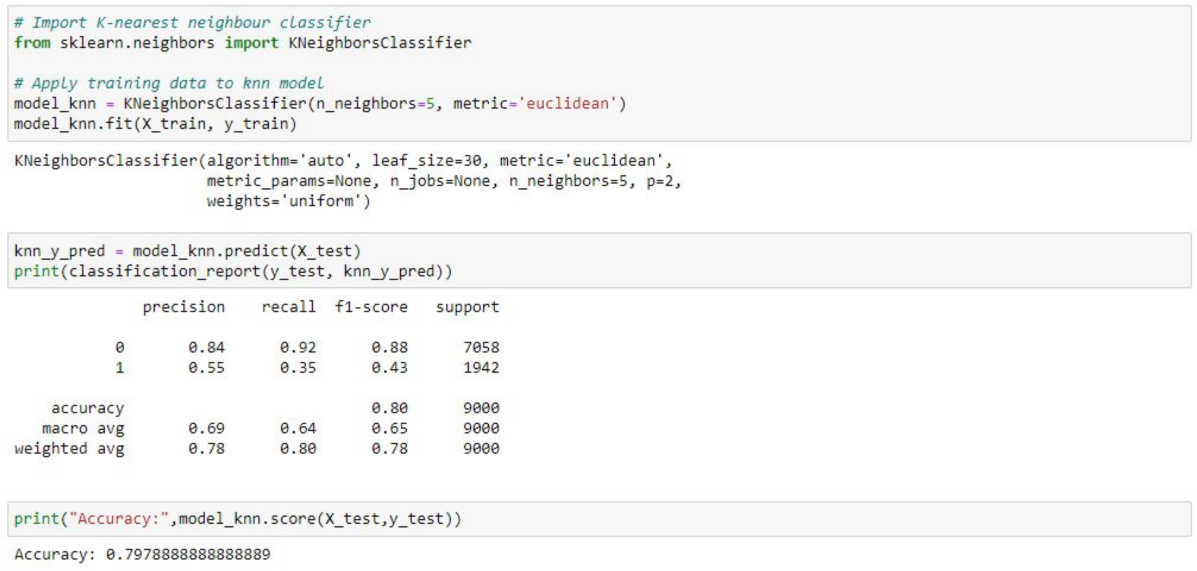
Figure 1.18 Creating test and train split



```
# Split data into 'X' features and 'y' target label sets (KNN and SVC)
X=ccd3[rfe_X_sug_cols].copy()
y=ccd3['NextPayment'].copy()
```

```
# Import module to split dataset
from sklearn.model_selection import train_test_split
# Split data set into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

Figure 1.19 Building the model



```
# Import K-nearest neighbour classifier
from sklearn.neighbors import KNeighborsClassifier

# Apply training data to knn model
model_knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
model_knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
knn_y_pred = model_knn.predict(X_test)
print(classification_report(y_test, knn_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      7058
           1       0.55      0.35      0.43      1942

    accuracy                           0.80      9000
   macro avg       0.69      0.64      0.65      9000
weighted avg       0.78      0.80      0.78      9000
```

```
print("Accuracy:",model_knn.score(X_test,y_test))
```

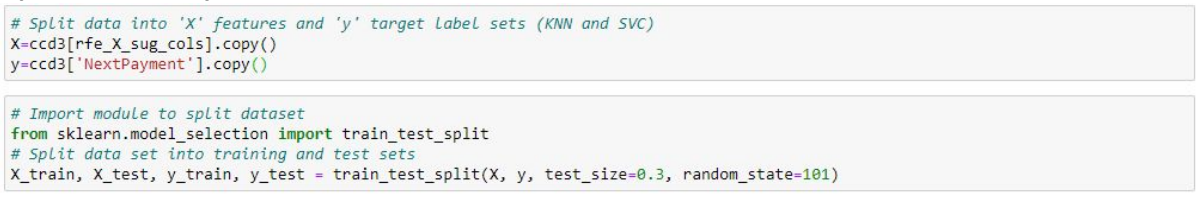
```
Accuracy: 0.7978888888888889
```

Compared to the Naive Bayes algorithm, we saw a similar accuracy score with KNN, coming in at 80%. Looking at the precision, we saw a slightly above average precision score of 55%. The recall for our KNN was worse compared to Naive Bayes - we only managed to get 35%, which resulted in a lower F1-Score of 43%. Using KNN as a model can be very helpful, as it does not work off assumptions in the data, and since it doesn't require any training steps, when new data is inputted it simply will label the new values based on its prior learning[7]. Some drawbacks to KNN is that it can become difficult to determine what the optimal K-factor should be when trying to classify new data and its sensitivity towards outliers[8].

[6] Tavish Srivastava, Analytics Vidhya, Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm (with implementation in Python & R), March 26, 2018 https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, Accessed December 19, 2019
[7] Genesis, Pros and Cons of K-Nearest Neighbours, September 25, 2018, https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/, Accessed December 20, 2019
[8] Genesis, Pros and Cons of K-Nearest Neighbours, September 25, 2018, https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/, Accessed December 20, 2019

SVM: SVM/SVC algorithm looks to plot each data point as a point in n-dimensional space (where n reflects the number of features we have input) with the value of each feature being the value of a particular coordinate, then it classifies by finding the hyperplane that differentiates the classes[9].

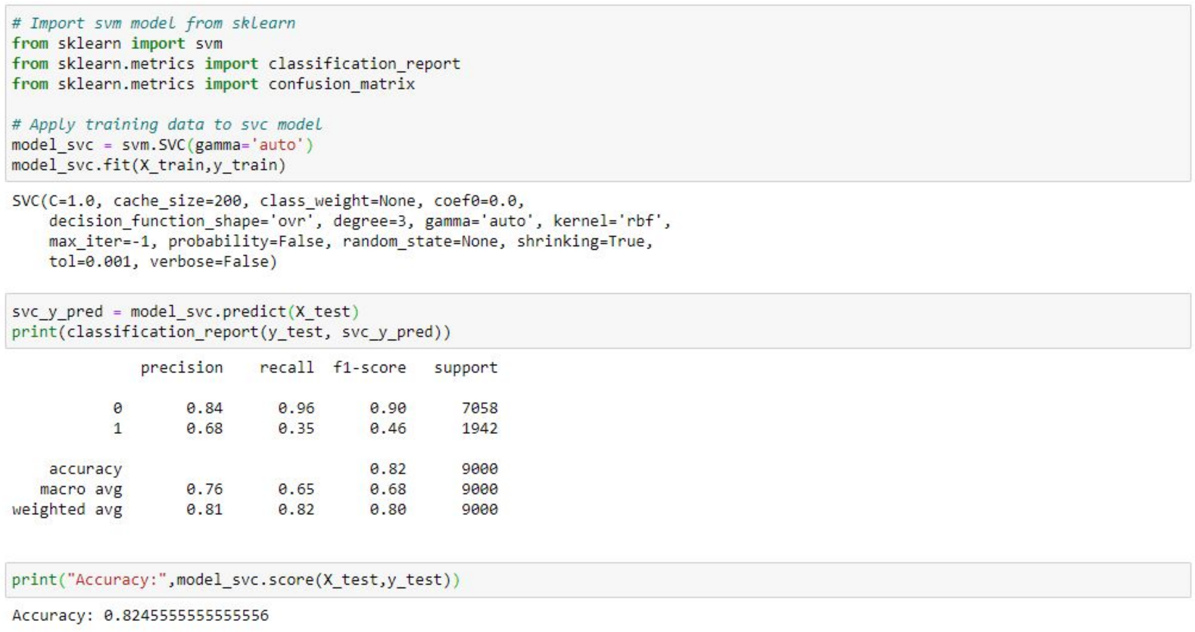
Figure 1.20 Creating test and train split



```
# Split data into 'X' features and 'y' target label sets (KNN and SVC)
X=ccd3[rfe_X_sug_cols].copy()
y=ccd3['NextPayment'].copy()
```

```
# Import module to split dataset
from sklearn.model_selection import train_test_split
# Split data set into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

Figure 1.21 Building the model



```
# Import svm model from sklearn
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Apply training data to svc model
model_svc = svm.SVC(gamma='auto')
model_svc.fit(X_train,y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
svc_y_pred = model_svc.predict(X_test)
print(classification_report(y_test, svc_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.96      0.90      7058
           1       0.68      0.35      0.46      1942

    accuracy                           0.82      9000
   macro avg       0.76      0.65      0.68      9000
weighted avg       0.81      0.82      0.80      9000
```

```
print("Accuracy:",model_svc.score(X_test,y_test))
```

```
Accuracy: 0.8245555555555556
```

Our SVM algorithm ended up with the highest accuracy of our three models, attaining 82%. We also saw the highest degree of precision with SVM, reaching a ratio 68% of correctly predicted defaulters (vs all predicted defaulters). The recall sensitivity was the same as we encountered with KNN, only reaching 35%. Overall, the F1-Score was 46%, just slightly below that of Naive Bayes. Utilizing an SVM/SVC model can be quite helpful, especially in instances with more outlier data, as it is much less sensitive to these compared to KNN[10]. A downside is that with very large datasets, it can take a longer time for the model to process which could lead to longer lead times for analysis[11].

[9] Sunil Ray, Analytics Vidhya, Understanding Support Vector Machine algorithm from examples (along with code), September 23, 2017, https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/, Accessed December 19, 2019
[10] Ajay Yadav, Towards Data Science, Support Vector Machines (SVM), October 20, 2018, https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589, Accessed December 20, 2019
[11] Ajay Yadav, Towards Data Science, Support Vector Machines (SVM), October 20, 2018, https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589, Accessed December 20, 2019

Evaluation / Findings

After reviewing each of the models, based on the balance of both accuracy and F1-Score, we could recommend implementing the SVM/SVC algorithm. While it came in with the second highest F1-Score, it did register the highest degree of accuracy just behind Naive Bayes. As a credit lender, the volume of client information that would be stored within their databases would be massive, so having an algorithm that can not only handle the volume, but can also achieve a higher degree of precision in determining correct positive results is critical, which is why we are recommending that of the models tested, SVM be implemented.

If cost were a major contributing factor in the decision making of the credit company, then utilizing Naive Bayes, although less precise that SVM, would be a good alternative as it would be much less costly to implement and still provides an equally high rate of model accuracy.

Conclusion / Suggestions

While all three of our classification models were able to meet our model criterion of an accuracy of at least 80%, none of them managed to reach out precision goal of at least 75%. For our Taiwanese credit lender, while having a model that has a high accuracy is important, precision of the model is likely even more important than just accuracy alone. If accuracy is thought of as how close to the actual value the model can get, then precision is essentially how capable the model is at reproducing the same results over and over.

In the case of our credit lender, the goal of implementing any machine-learning algorithm is to be able to consistently and correctly identify which customers are on the brink of defaulting. For our credit company, there is a very high business cost associated with incorrectly classifying customers as potential defaulters - this could lead to them chasing down customers who aren't the ones who will cost you money down the line and make you blind to those who will be the biggest financial threat to your business’ bottom line.

As a credit lender, the company has to do its due diligence to not only ensure that they are properly assessing potential credit customers to understand their ability to repay, but also to monitor the level of credit that its customers are utilizing and ensuring that it has proper policies in place to intervene early in the repayment process if it appears that a customer may be having difficulty consistently paying back their loan.

Much of the financial crisis that occurred in Taiwan back in the mid-aughts could have been mitigated if the banks had done a better job of educating their potential customers about safe credit practices, implemented stricter screening requirements and had better systems in place to not only identify potential at-risk customers (leveraging machine-learning to help identify patterns early on) but also to outline how to work with customers to find ways to help them repay their debts over time, rather than let the owing sums balloon to the point of causing large-scale societal problems, some of which are still being dealt with today.

As a next step, we would suggest going back to the SVM model and continue to refine it to help raise the level of precision and recall before presenting our final recommendation to the Taiwanese company. As our dataset was only looking at a small subset of customers (30,000), we would recommend re-running the model with a much larger dataset to help the test/train and compare and evaluate the results against our initial output to uncover where the model can be improved.

Works Cited

1. Canadian Bankers Association, *Credit Card Delinquency - VISA and Mastercard*, November 26, 2019, https://cba.ca/credit-card-delinquency-and-loss-statistics, Accessed December 9, 2019.

2. Liang Yu et al., S&P Global, *Will China's Credit Card Boom Follow the Well-worn Path to Bust?*, July 4, 2019, https://www.spglobal.com/en/research-insights/articles/will-china-s-credit-card-boom-follow-the-well-worn-path-to-bust, Accessed December 10, 2019

3. Eric Wang, Seven Pillars Institute, *The Taiwan Credit Card Crisis*, https://sevenpillarsinstitute.org/case-studies/taiwans-credit-card-crisis/, Accessed December 14, 2019

4. Sunil Ray, Analytics Vidhya, *6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R*, September 11, 2017, https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/, Accessed December 18, 2019

5. Tavish Srivastava, Analytics Vidhya, *Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm (with implementation in Python & R)*, March 26, 2018 https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, Accessed December 19, 2019

6. Genesis, *Pros and Cons of K-Nearest Neighbours, September 25, 2018*, https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/, Accessed December 20, 2019

7. Sunil Ray, Analytics Vidhya, *Understanding Support Vector Machine algorithm from examples (along with code)*, September 23, 2017, https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/, Accessed December 19, 2019

8. Ajay Yadav, Towards Data Science, *Support Vector Machines (SVM)*, October 20, 2018, https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589, Accessed December 20, 2019