Best Marriage Partner

A table of Contents

[Classification]

Classification

For Wedding Information Company

“Best Marriage Partner”

To increase the success rate of marriage
and the satisfaction of a customer
in a marriage information company.

2 Data set

Data Source :

http://archive.ics.uci.edu/ml/datasets/Bank+Marketing#

- ✓ There are 16 predictors and class (y or n)
- ✓ We will use attributes related to the marriage

  ex) Age, job, marital, education, debt, loan,

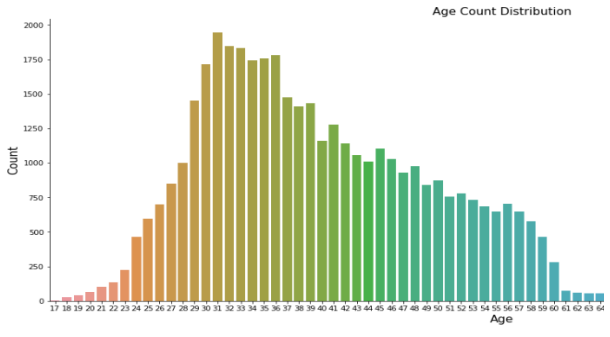
Data Type and Dirty Data

- ✓ This data set is composed of categorical data(job), numerical data(age), etc.
- ✓ need to make dirty data like missing / Wrong / Unusable data, Outliers.

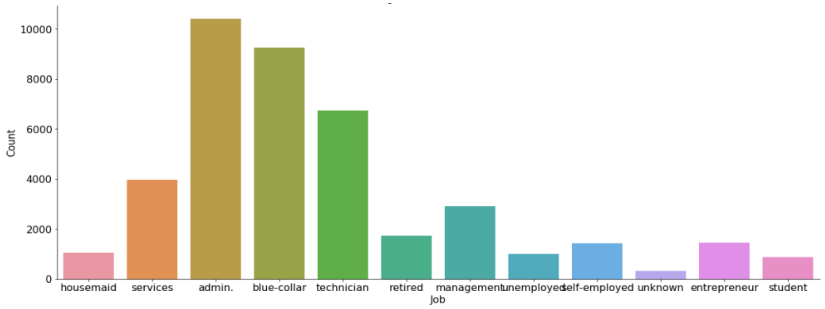
Statistical Description of Data set

[Data Distribution]

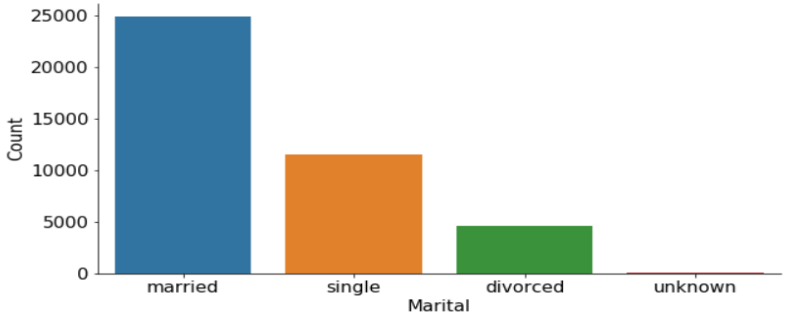
- Age



- Job



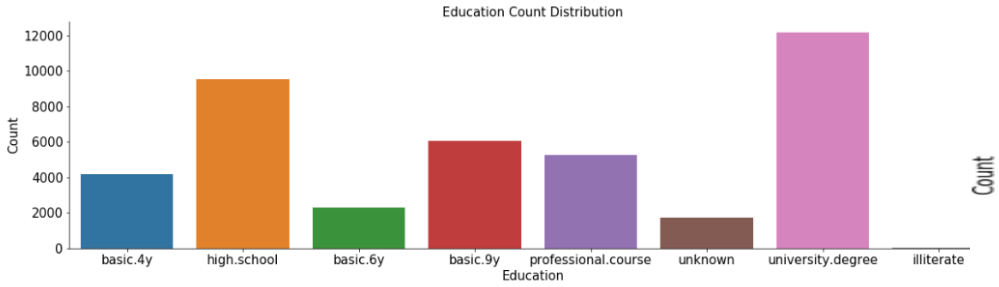
- Marital



Statistical Description of Dataset

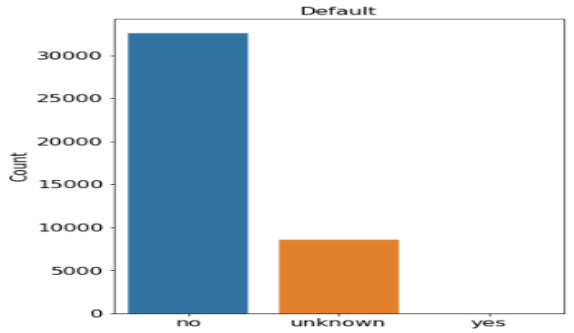
[Data Distribution]

- Education



- Loan

- Debt



Missing Data

Restructuring

[Table Vertical Decomposition]

: Drop all the features that look unrelated to marriage.

```
columns = ['age', 'job', 'marital', 'education', 'default', 'housing', 'loan','balance']
df = df[columns]
df = df.astype({"age":int,"balance": float})
```

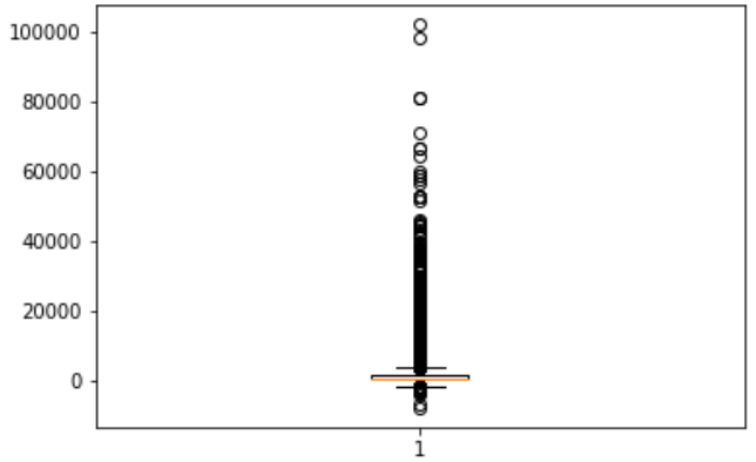
- 16 features → 8 features

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'balance'],
```

Data Value Changes

[Cleaning Dirty Data]

: Handle Missing data, outliers, redundancy



Missing Data

```
# Missing Data -> Change "unknown" to NaN
df = df.replace('unknown',np.nan)
print( * Missing data before removal *")
print("Total data : ",len(df))
df.isnull().sum()
df = df.dropna() # Delete Missing Data
print("* Missing data after removal *")
print("Total data : ",len(df))
df.isnull().sum() # Check Missing Data
```

Outliers

```
df = df[df["balance"]<=40000]
df = df[df["balance"]>=-500]
```

Redundancy

```
# Delete Redundancy
df = df.drop_duplicates()
```

Results (45211 → 38476)

```
* Missing data before removal *
Total data :  45211
* Missing data after removal *
Total data :  43193
* data before removing redundancy * 42498
* data after removing redundancy * 38476
```

Convert ‘unknown’ into NaN

Data Value Changes

[Normalization]

: Data is normalized to a number between 0 and 1 to use algorithm

MinMax Scaler

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
feature = scaler.fit_transform(feature)
```

Before

	age	job	education	default	housing	loan	balance
0	58	4	2	0	1	0	2143.0
1	44	9	1	0	1	0	29.0
2	33	2	1	0	1	1	2.0
5	35	4	2	0	1	0	231.0
6	28	4	2	0	1	1	447.0
7	42	2	2	1	1	0	2.0

After

```
[[0.51948052 0.4        1.         ... 1.         0.         0.06626551]
 [0.33766234 0.9        0.5        ... 1.         0.         0.01326313]
 [0.19480519 0.2        0.5        ... 1.         1.         0.01258619]
 ...
 [0.68831169 0.5        0.         ... 0.         0.         0.05588567]
 [0.50649351 0.1        0.5        ... 0.         0.         0.02928419]
 [0.24675325 0.2        0.5        ... 0.         0.         0.0870252 ]]
```

Feature Engineering

[Feature Creation] - LabelEncoding

: Converts categorical data into integers

```
# Label Encoding
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
labelEncoder.fit(df['job'])
df['job'] = labelEncoder.transform(df['job'])

labelEncoder.fit(df['marital'])
df['marital'] = labelEncoder.transform(df['marital'])

labelEncoder.fit(df['education'])
df['education'] = labelEncoder.transform(df['education'])

labelEncoder.fit(df['default'])
df['default'] = labelEncoder.transform(df['default'])

labelEncoder.fit(df['housing'])
df['housing'] = labelEncoder.transform(df['housing'])

labelEncoder.fit(df['loan'])
df['loan'] = labelEncoder.transform(df['loan'])
```

Example

Married → 1
Single → 2

Before

	age	job	marital	education	default	housing	loan	balance
0	58	management	married	tertiary	no	yes	no	2143.0
1	44	technician	single	secondary	no	yes	no	29.0
2	33	entrepreneur	married	secondary	no	yes	yes	2.0
5	35	management	married	tertiary	no	yes	no	231.0

After

	age	job	education	default	housing	loan	balance
0	58	4	2	0	1	0	2143.0
1	44	9	1	0	1	0	29.0
2	33	2	1	0	1	1	2.0
5	35	4	2	0	1	0	231.0

0	1
1	2
2	1
5	1

Feature Engineering

[Feature Reduction] - PCA

: Extracts major k components with a larger data variance.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=4)
x_train = pca.fit_transform(x_train)
```

Before

```
[[0.19480519 0.2        0.5        ... 0.         1.         0.01118215]
 [0.48051948 0.4        1.         ... 0.         0.         0.02615018]
 [0.22077922 0.4        1.         ... 1.         0.         0.08745142]
```

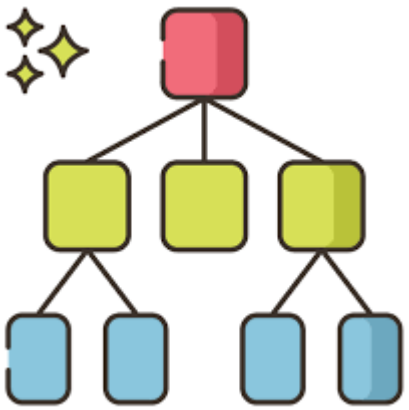
After

* 7 → 4 columns

```
[[ 0.48493311  0.92743166 -0.03877083 -0.15217892]
 [ 0.61071227 -0.18962543  0.16146155 -0.31662943]
 [-0.38356811 -0.27717319  0.3296693  -0.2791112 ]
```

Decision Tree

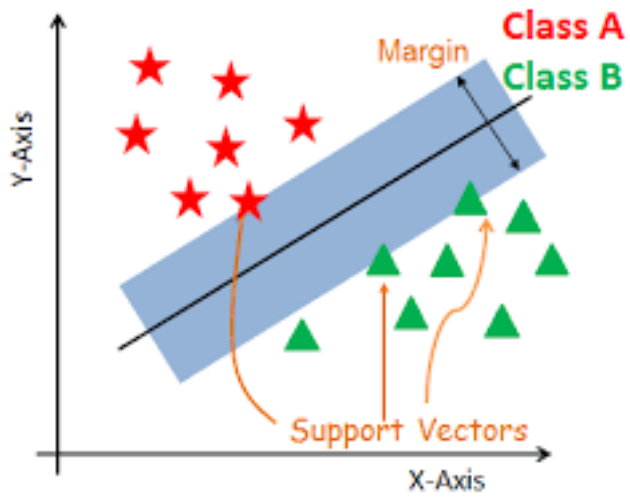
→ Use 'GridSearchCV' to find the best score and parameter



```
decision_param = {'criterion' : ['gini', 'entropy'],
                  'min_samples_split' : [2, 10, 20],
                 'max_depth' : [1, 5, 10, 15],
                  'min_samples_leaf' : [1, 5, 10],
                  'max_leaf_nodes' : [5, 10, 20, 30, 40, 50],
                 'random_state' : [30, 50, 100]}

# Build three classifier
decision = DecisionTreeClassifier()

# Bulid GridSearchCV which find the parameter combination with the highest score
grid_decision = GridSearchCV(decision, param_grid = decision_param, scoring = 'accuracy', cv = 10)

# Training Decision Tree GridSearchCV
grid_decision.fit(x_train, y_train)
```

SVM

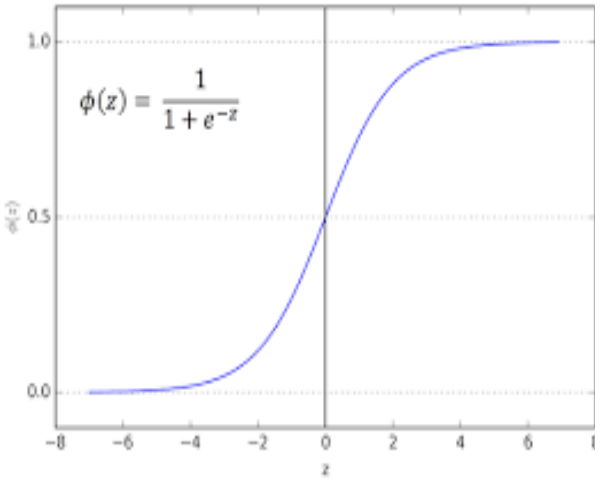
→ Use 'GridSearchCV' to find the best score and parameter




```
svm_param = {'kernel' : ['linear','poly','rbf','sigmoid'],
                'gamma' : [10, 100],
                'C' : [0.1, 1.0, 10.0]}

# Build svm classifier
svm = SVC()

# Bulid GridSearchCV which find the parameter combination with the highest score
grid_svm = GridSearchCV(svm, param_grid = svm_param, scoring = 'accuracy', cv = 10)

# Training Decision Tree GridSearchCV
grid_svm.fit(x_train, y_train)
```

Logistic Regression

→ Use 'GridSearchCV' to find the best score and parameter




```
# Logistic Regression Parameter
logistic_param = {'solver' : ['liblinear', 'lbfgs', 'sag'],
                  'max_iter' : [50, 100, 200]}

# Build logisticRegression classifier
logistic = LogisticRegression()

# Bulid GridSearchCV which find the parameter combination with the highest score
grid_logistic = GridSearchCV(logistic, param_grid = logistic_param, scoring = 'accuracy', cv = 10)

# Training Decision Tree GridSearchCV
grid_logistic.fit(x_train, y_train)
```

6 Data Evaluation

[Decision Tree]

```
* Decision Tree *
Best Parameter : DecisionTreeClassifier(class_weight=None, criterion='gini', max_dep
th=10,
            max_features=None, max_leaf_nodes=40,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=30,
            splitter='best')
High Score : 0.666617161103479
```

[Logistic Regression]

```
* Logistic Regression *
Best Parameter : LogisticRegression(C=1.0, class_weight=None, dual=False, f
          intercept_scaling=1, max_iter=50, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
High Score : 0.6634611814502654
```

[SVM]

```
* SVM *
Best Parameter : SVC(C=10.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=100, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
High Score : 0.6583373556603423
```

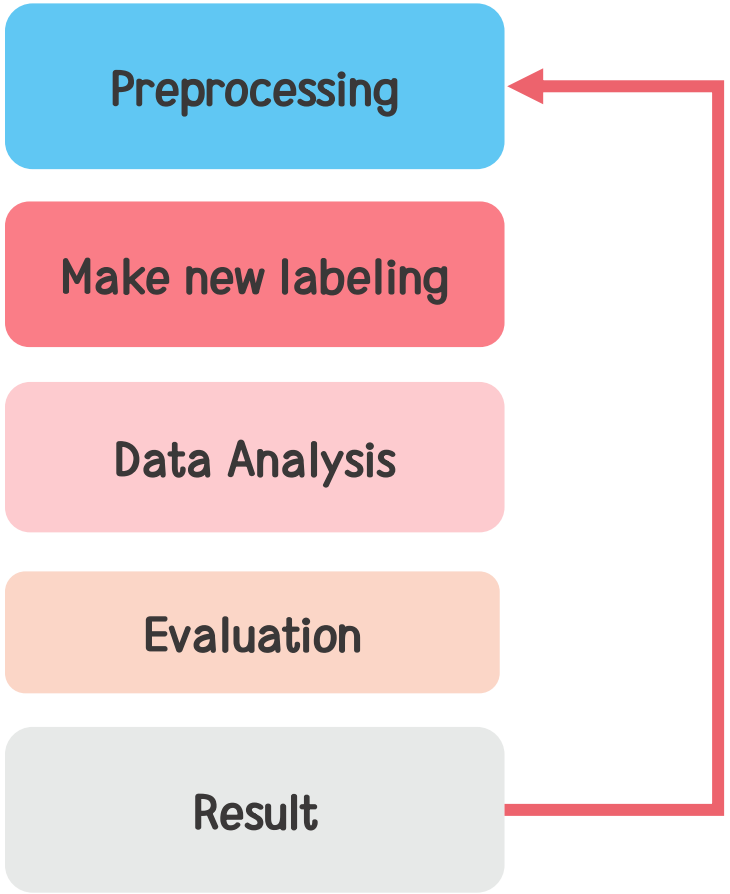
Score : 0.66
The score is too low

We had to go back
to the previous process.

7 Back to preprocessing

[End to End Big Data Process]

1. Objective Setting
2. Data Curation
3. Data Inspection
4. Data Preprocessing
5. Data Analysis
6. Evaluation
7. Deployment



3 Data Preprocessing & Data Analysis

We did the following to raise the score

- ✓ Use standard scaler
- ✓ Delete pca or change the number of component
- ✓ Change algorithm → Random Forest Classification

→ But, the results have not changed

Reduce the number of classes

3 class

Married / Single / Divorced

2 class

Married or Single

→ The result has improved.

2 Data Evaluation

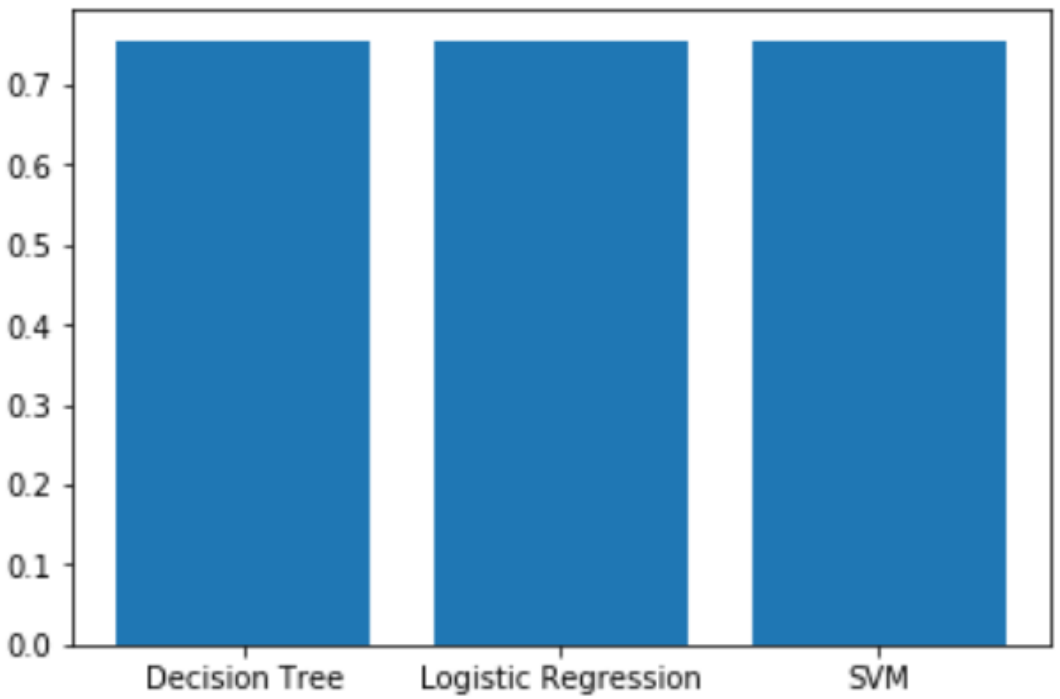
```
* Decision Tree *
Best Parameter : DecisionTreeClassifier(class_weight=None,
            max_features=None, max_leaf_nodes=40,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, r
            splitter='best')
High Score : 0.7551106427818757
```

Score : 0.66

Score : 0.75

```
* Logistic Regression *
Best Parameter : LogisticRegression(C=1.0, class_weight=N
          intercept_scaling=1, max_iter=50, multi_class='
          penalty='l2', random_state=None, solver='sag',
          verbose=0, warm_start=False)
High Score : 0.7555321390937829
```

```
* SVM *
Best Parameter : SVC(C=1.0, cache_size=200, class_weight=
  decision_function_shape='ovr', degree=3, gamma=100, ker
  max_iter=-1, probability=False, random_state=None, shri
  tol=0.001, verbose=False)
High Score : 0.7555742887249737
```



2 Data Evaluation

Confusion Matrix

[Decision Tree]

```
[[6108  771]
 [1783 1507]]
```

[Logistic Regression]

```
[[5939  940]
 [1599 1691]]
```

[SVM]

```
[[6072  807]
 [1698 1592]]
```

** Decision Tree **

	precision	recall	f1-score	support
class 0	0.77	0.89	0.83	6879
class 1	0.66	0.46	0.54	3290
avg / total	0.74	0.75	0.73	10169

** Logistic Regression **

	precision	recall	f1-score	support
class 0	0.79	0.86	0.82	6879
class 1	0.64	0.51	0.57	3290
avg / total	0.74	0.75	0.74	10169

** SVM **

	precision	recall	f1-score	support
class 0	0.78	0.88	0.83	6879
class 1	0.66	0.48	0.56	3290
avg / total	0.74	0.75	0.74	10169

Thank you