CSC4008 Assignment 9

Jiang Jingxin 117020119

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

0. Credit Dataset

In [2]:
```python
raw_data = pd.read_csv("credit-g.csv")
```

In [3]:
```python
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   checking_status         1000 non-null   object
 1   duration                1000 non-null   int64 
 2   credit_history          1000 non-null   object
 3   purpose                 1000 non-null   object
 4   credit_amount           1000 non-null   int64 
 5   savings_status          1000 non-null   object
 6   employment              1000 non-null   object
 7   installment_commitment  1000 non-null   int64 
 8   personal_status         1000 non-null   object
 9   other_parties           1000 non-null   object
 10  residence_since         1000 non-null   int64 
 11  property_magnitude      1000 non-null   object
 12  age                     1000 non-null   int64 
 13  other_payment_plans     1000 non-null   object
 14  housing                 1000 non-null   object
 15  existing_credits        1000 non-null   int64 
 16  job                     1000 non-null   object
 17  num_dependents          1000 non-null   int64 
 18  own_telephone           1000 non-null   object
 19  foreign_worker          1000 non-null   object
 20  class                   1000 non-null   object
dtypes: int64(7), object(14)
memory usage: 164.2+ KB
```

In [4]:
```python
raw_data.nunique()
```

Out[4]:
```
checking_status             4
duration                   33
credit_history              5
purpose                    10
credit_amount             921
savings_status              5
employment                  5
installment_commitment      4
personal_status             4
other_parties               3
residence_since             4
property_magnitude          4
age                        53
other_payment_plans         3
housing                     3
existing_credits            4
job                         4
num_dependents              2
own_telephone               2
foreign_worker              2
class                       2
dtype: int64
```

In [5]:
```python
X = raw_data.drop(columns=['class'])
y = raw_data.loc[:,'class']
```

1. Naïve Bayes Classifier

In [311]:
```python
class NaiveBayes:
    def __init__(self, pseudocount=1):
        self.pseudocount = pseudocount

    def fit(self, X, y, numeric):
        self._class = y.unique()
        self._numeric_idx = list()
        for i, col in enumerate(X.columns):
            if numeric==[]:
                break
            if col in numeric:
                self._numeric_idx.append(i)
                numeric.remove(col)
        self._prior = dict()
        self._theta = list()
        for label in self._class:
            self._prior[label] = sum(y==label)/len(y)
            x = X[y==label]
            theta = [dict()for i in range(len(X.columns))]
            i = 0
            for idx in self._numeric_idx:
                while i < idx:
                    for attr in X.iloc[:,i].unique():
                        theta[i][attr] = (sum(x.iloc[:,i]==attr)+1)/(len(x)+len(X.iloc[:,i].unique()))
                    i += 1
                theta[i]['mu'] = x.iloc[:,i].mean()
                theta[i]['sigma'] = x.iloc[:,i].std()
                i += 1
            while i < len(X.columns):
                for attr in X.iloc[:,i].unique():
                    theta[i][attr] = (sum(x.iloc[:,i]==attr)+1)/(len(x)+len(X.iloc[:,i].unique()))
                i += 1
            self._theta.append(theta)

    def predict(self,X):
        if len(self._numeric_idx)>0:
            from scipy.stats import norm
        pred = list()
        for row in range(len(X)):
            x = X.loc[row]
            Label = ''
            Posterior = 0
            for l, label in enumerate(self._class):
                posterior = self._prior[label]
                i = 0
                for idx in self._numeric_idx:
                    while i < idx:
                        posterior *= self._theta[l][i][x[i]]
                        i += 1
                    posterior *= norm.pdf(x[i],self._theta[l][i]['mu'],self._theta[l][i]['sigma'])
                    i += 1
                while i < len(X.columns):
                    posterior *= self._theta[l][i][x[i]]
                    i += 1
                if posterior > Posterior:
                    Posterior = posterior
                    Label = label
            pred.append(Label)
        pred = pd.Series(pred,name='predicted')
        return pred
```

1.1 Cross Validation

In [312]:
```python
def cv(X, y, k=10, seed =None, numeric=['credit_amount','age','duration']):
    from tqdm import tqdm
    np.random.seed(seed)

    idx = np.arange(y.size)
    np.random.shuffle(idx)
    fold = np.array_split(idx,k) # split shuffled index into k folds

    pred = np.zeros_like(y)

    # cross validation using k folds
    for i in tqdm(range(k)):
        test_idx = fold[i]
        train_idx = np.setdiff1d(idx, test_idx)
        mod = NaiveBayes()
        mod.fit(X.loc[train_idx],y[train_idx],numeric=numeric.copy())
        pred[test_idx] = mod.predict(X.loc[test_idx].reset_index(drop=True))
    return pd.Series(pred,name='predicted')
```

1.2 Performance Evaluation

In [313]:
```python
def evaluate(true,pred):
    cm = pd.crosstab(true,pred)
    TP = cm.iloc[1,1]
    TN = cm.iloc[0,0]
    FP = cm.iloc[0,1]
    FN = cm.iloc[1,0]
    precision = round(TP/ (TP+FP),3)
    sensitivity = round(TP / (TP + FN), 3)
    specificity = round(TN / (TN + FP), 3)
    F_measure = round(2*precision*sensitivity/(precision + sensitivity),3)
    print("\n=== Detailed Accuracy ===\n")
    print("Precision:", precision,sep='\t')
    print("Sensitivity:", sensitivity,sep='\t')
    print("Specificity:", specificity,sep='\t')
    print("F_measure:", F_measure,sep='\t')
    print("\n=== Confusion Matrix ===\n")
    print(cm)
```

1.3 Results

In [314]:
```python
pred = cv(X,y,seed=1)
evaluate(y,pred)
```

```
100%|██████████| 10/10 [00:04<00:00,  2.25it/s]

=== Detailed Accuracy ===

Precision:      0.801
Sensitivity:    0.866
Specificity:    0.497
F_measure:      0.832

=== Confusion Matrix ===

predicted  bad  good
class
bad        149   151
good        94   606
```

2. Comparison

2.1 Compare with Weka

In [15]:
```python
import weka.core.jvm as jvm
jvm.start()
```

```
DEBUG:weka.core.jvm:Adding bundled jars
DEBUG:weka.core.jvm:Classpath=['/Users/Jingxin/.pyenv/versions/3.6.10/envs/virtual/lib/python3.6/site-packages/javabridge/jars/rhino-1.7R4.jar', '/Users/Jingxin/.pyenv/versions/3.6.10/envs/virtual/lib/python3.6/site-packages/javabridge/jars/runnablequeue.jar', '/Users/Jingxin/.pyenv/versions/3.6.10/envs/virtual/lib/python3.6/site-packages/javabridge/jars/cpython.jar', '/Users/Jingxin/.pyenv/versions/virtual/lib/python3.6/site-packages/weka/lib/python-weka-wrapper.jar', '/Users/Jingxin/.pyenv/versions/virtual/lib/python3.6/site-packages/weka/lib/weka.jar']
DEBUG:weka.core.jvm:MaxHeapSize=default
DEBUG:weka.core.jvm:Package support disabled
```

In [270]:
```python
from weka.core.converters import Loader
loader = Loader(classname="weka.core.converters.ArffLoader")
arff_data = loader.load_file("credit-g.arff")
arff_data.class_is_last()
```

In [275]:
```python
from weka.classifiers import Classifier, Evaluation
cls = Classifier(classname="weka.classifiers.bayes.NaiveBayes")
cls.build_classifier(arff_data)
```

In [276]:
```python
from weka.core.classes import Random
evaluation = Evaluation(arff_data)
evl = evaluation.crossvalidate_model(cls, arff_data, 10, Random(1))
print(evaluation.summary())
print(evaluation.class_details())
print(evaluation.matrix())
```

```
Correctly Classified Instances         754               75.4    %
Incorrectly Classified Instances       246               24.6    %
Kappa statistic                          0.3813
Mean absolute error                      0.2936
Root mean squared error                  0.4201
Relative absolute error                 69.8801 %
Root relative squared error             91.6718 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.864    0.503    0.800      0.864    0.831      0.385    0.787     0.891     good
                 0.497    0.136    0.611      0.497    0.548      0.385    0.787     0.577     bad
Weighted Avg.    0.754    0.393    0.743      0.754    0.746      0.385    0.787     0.797

=== Confusion Matrix ===

   a   b   <-- classified as
 605  95 |   a = good
 151 149 |   b = bad
```

In [8]:
```python
jvm.stop()
```

2.2 Different k

2.2.1 k = 2

In [261]:
```python
pred = cv(X,y,k=2, seed=1)
evaluate(y,pred)
```

```
100%|██████████| 2/2 [00:03<00:00,  1.95s/it]

=== Detailed Accuracy ===

Precision:      0.785
Sensitivity:    0.827
Specificity:    0.47
F_measure:      0.805

=== Confusion Matrix ===

predicted  bad  good
class
bad        141   159
good       121   579
```

2.2.2 k = 5

In [256]:
```python
pred = cv(X,y,k=5, seed=1)
evaluate(y,pred)
```

```
100%|██████████| 5/5 [00:03<00:00,  1.43it/s]

=== Detailed Accuracy ===

Precision:      0.796
Sensitivity:    0.86
Specificity:    0.487
F_measure:      0.827

=== Confusion Matrix ===

predicted  bad  good
class
bad        146   154
good        98   602
```

2.2.3 k = 10

In [315]:
```python
pred = cv(X,y,k=10, seed=1)
evaluate(y,pred)
```

```
100%|██████████| 10/10 [00:05<00:00,  1.91it/s]

=== Detailed Accuracy ===

Precision:      0.801
Sensitivity:    0.866
Specificity:    0.497
F_measure:      0.832

=== Confusion Matrix ===

predicted  bad  good
class
bad        149   151
good        94   606
```

2.2.4 k = 20

In [260]:
```python
pred = cv(X,y,k=20, seed=1)
evaluate(y,pred)
```

```
100%|██████████| 20/20 [00:06<00:00,  3.18it/s]

=== Detailed Accuracy ===

Precision:      0.802
Sensitivity:    0.873
Specificity:    0.497
F_measure:      0.836

=== Confusion Matrix ===

predicted  bad  good
class
bad        149   151
good        89   611
```

2.2.4 k = 50

In [262]:
```python
pred = cv(X,y,k=50, seed=1)
evaluate(y,pred)
```

```
100%|██████████| 50/50 [00:09<00:00,  5.55it/s]

=== Detailed Accuracy ===

Precision:      0.799
Sensitivity:    0.866
Specificity:    0.493
F_measure:      0.831

=== Confusion Matrix ===

predicted  bad  good
class
bad        148   152
good        94   606
```

2.2.5 Summary

As k increases, the cross-validated performance first increases with k.

When k gets larger (e.g. k=50), it may suffer from overfitting problem and the performance is not so good as before.

2.3 Selection of Attributes

2.3.1 CfsSubsetEval (BestFirst)

In [277]:
```python
from weka.filters import Filter
from weka.attribute_selection import ASSearch, ASEvaluation

flter = Filter(classname="weka.filters.supervised.attribute.AttributeSelection")
aseval = ASEvaluation(classname="weka.attributeSelection.CfsSubsetEval",
                      options=["-P", "1", "-E", "1"])
assearch = ASSearch(classname="weka.attributeSelection.BestFirst",
                    options=["-D", "1", "-N", "5"])
flter.set_property("evaluator", aseval.jobject)
flter.set_property("search", assearch.jobject)
flter.inputformat(arff_data)
filtered = flter.filter(arff_data)
print(filtered.summary(filtered))
```

```
Relation Name:  german_credit-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.CfsSubset
Eval -P 1 -E 1-Sweka.attributeSelection.BestFirst -D 1 -N 5
Num Instances:  1000
Num Attributes: 4

     Name                        Type  Nom  Int Real     Missing      Unique  Dist
1 checking_status             Nom 100%   0%   0%     0 /  0%     0 /  0%     4
2 duration                    Num   0% 100%   0%     0 /  0%     5 /  1%    33
3 credit_history              Nom 100%   0%   0%     0 /  0%     0 /  0%     5
4 class                       Nom 100%   0%   0%     0 /  0%     0 /  0%     2
```

In [286]:
```python
X_filtered = X[["checking_status","duration","credit_history"]]
```

In [303]:
```python
pred = cv(X_filtered,y,k=50, seed=1,numeric=["duration"])
evaluate(y,pred)
```

```
100%|██████████| 50/50 [00:02<00:00, 19.59it/s]

=== Detailed Accuracy ===

Precision:      0.764
Sensitivity:    0.91
Specificity:    0.343
F_measure:      0.831

=== Confusion Matrix ===

predicted  bad  good
class
bad        103   197
good        63   637
```

2.3.2 CorrelationAttributeEval (Ranker)

In [317]:
```python
from weka.filters import Filter
from weka.attribute_selection import ASSearch, ASEvaluation

flter = Filter(classname="weka.filters.supervised.attribute.AttributeSelection")
aseval = ASEvaluation(classname="weka.attributeSelection.CorrelationAttributeEval")
assearch = ASSearch(classname="weka.attributeSelection.Ranker",
                    options=["-T", "-1.7976931348623157E308", "-N", "-1"])
flter.set_property("evaluator", aseval.jobject)
flter.set_property("search", assearch.jobject)
flter.inputformat(arff_data)
filtered = flter.filter(arff_data)
print(filtered.summary(filtered))
```

```
Relation Name:  german_credit-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.Correlati
onAttributeEval-Sweka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Num Instances:  1000
Num Attributes: 21

     Name                        Type  Nom  Int Real     Missing      Unique  Dist
 1 checking_status            Nom 100%   0%   0%     0 /  0%     0 /  0%     4
 2 duration                   Num   0% 100%   0%     0 /  0%     5 /  1%    33
 3 credit_amount              Num   0% 100%   0%     0 /  0%   847 / 85%   921
 4 savings_status             Nom 100%   0%   0%     0 /  0%     0 /  0%     5
 5 housing                    Nom 100%   0%   0%     0 /  0%     0 /  0%     3
 6 other_payment_plans        Nom 100%   0%   0%     0 /  0%     0 /  0%     3
 7 age                        Num   0% 100%   0%     0 /  0%     1 /  0%    53
 8 credit_history             Nom 100%   0%   0%     0 /  0%     0 /  0%     5
 9 foreign_worker             Nom 100%   0%   0%     0 /  0%     0 /  0%     2
10 purpose                    Nom 100%   0%   0%     0 /  0%     0 /  0%    10
11 installment_commitment     Num   0% 100%   0%     0 /  0%     0 /  0%     4
12 personal_status            Nom 100%   0%   0%     0 /  0%     0 /  0%     4
13 property_magnitude         Nom 100%   0%   0%     0 /  0%     0 /  0%     4
14 employment                 Nom 100%   0%   0%     0 /  0%     0 /  0%     5
15 existing_credits           Num   0% 100%   0%     0 /  0%     0 /  0%     4
16 own_telephone              Nom 100%   0%   0%     0 /  0%     0 /  0%     2
17 job                        Nom 100%   0%   0%     0 /  0%     0 /  0%     4
18 other_parties              Nom 100%   0%   0%     0 /  0%     0 /  0%     3
19 num_dependents             Num   0% 100%   0%     0 /  0%     0 /  0%     2
20 residence_since            Num   0% 100%   0%     0 /  0%     0 /  0%     4
21 class                      Nom 100%   0%   0%     0 /  0%     0 /  0%     2
```

In [318]:
```python
X_filtered = X[["checking_status","duration","credit_amount","savings_status","housing"]] # top 5
```

In [319]:
```python
pred = cv(X_filtered,y,k=50, seed=1,numeric=["duration","credit_amount"])
evaluate(y,pred)
```

```
100%|██████████| 50/50 [00:03<00:00, 16.00it/s]

=== Detailed Accuracy ===

Precision:      0.759
Sensitivity:    0.889
Specificity:    0.34
F_measure:      0.819

=== Confusion Matrix ===

predicted  bad  good
class
bad        102   198
good        78   622
```

2.3.3 Summary

The attributes selection is a trading off process: fewer predictor and less information, more predictor and more noise in data.

After selection, the sensitivity increase while other measures drops.

However, even with smaller set of predictors (3 of 20), the results are roughly the same.

3. Bagging

In [304]:
```python
def bagging(X,y,k,seed=None,numeric = ['credit_amount','age','duration']):
    from tqdm import tqdm
    np.random.seed(seed)
    pred = np.zeros_like(y)
    for i in tqdm(range(k)):
        idx = np.random.choice(len(X),len(X),replace=True)
        X_train = X.loc[idx].reset_index(drop=True)
        y_train = y.loc[idx].reset_index(drop=True)
        mod = NaiveBayes()
        mod.fit(X_train,y_train,numeric=numeric.copy())
        pred += (mod.predict(X)=="good")
    pred = pd.Series(np.where(pred > k//2,'good','bad'),name='predicted')
    return pred
```

In [308]:
```python
pred = bagging(X,y,k=10,seed=1)
evaluate(y,pred)
```

```
100%|██████████| 10/10 [00:37<00:00,  3.76s/it]

=== Detailed Accuracy ===

Precision:      0.814
Sensitivity:    0.873
Specificity:    0.533
F_measure:      0.842

=== Confusion Matrix ===

predicted  bad  good
class
bad        160   140
good        89   611
```

After using the bagging strategy, the results improve quite a lot with higher accuracy comparing to the original one.

This is because the composite model reduces the variance of individual errors.