Report

yuweic3

2017/11/22

Section 1

Python

Step 1: View data with Python. Please refer to the second section of Python part.

Here is the information about data:

There are 1727 items, 7 attributes and one used for classification.

For classification attributes, there 384 were acceptable, 69 (4.05 %) were good, 1210 (69.85 %) were unacceptable, and 65 cars (3.82%) were very good.

Step 2: Preprocessing data by converting nominal attribute to numeric attribute

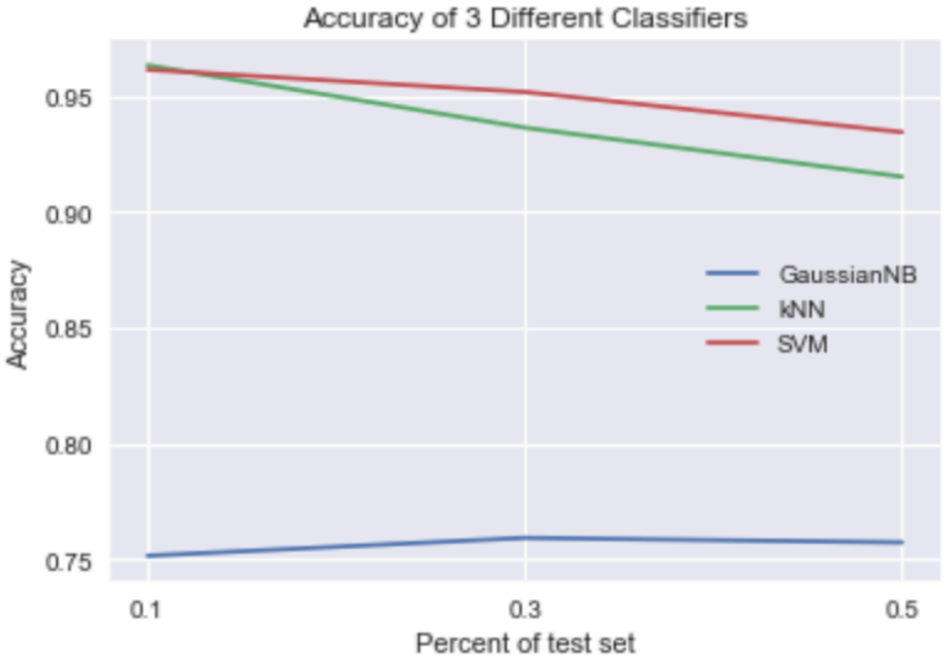
For example, convert 'high' to 3.

Step 3. Classifying

Here I use three classifiers: Naïve Bayes, kNN and SVM. And I use accuracy for evaluation of three algorithms and split the original dataset with three different percent [10%, 30% 50%].

Accuracy of three different classifiers

Classifier\test percent	10%	30%	50%
GaussianNB	0.751	0.759	0.757
SVM	0.961	0.952	0.934
kNN	0.963	0.936	0.915



It is clear that SVM has best performance while Gaussian Naive Bayes has the worst performance.

Weka:

Here I used car csv dataset processed by Python. Similarly, I use Naïve Bayes, kNN and SVM classifiers.

First preprocessing by converting numeric class attribute into nominal type.

Then apply classifications:

1. NaiveBayes: [weka.classifiers.bayes.NaiveBayes]

   10% test: (Accuracy : 83.66%)

```
=== Summary ===

Correctly Classified Instances        1301               83.6656 %
Incorrectly Classified Instances       254               16.3344 %
Kappa statistic                          0.666
Mean absolute error                      0.1193
Root mean squared error                  0.2343
Relative absolute error                 52.8026 %
Root relative squared error             68.9768 %
Total Number of Instances             1555
```

   30% test: (Accuracy : 83.38%)

```
=== Summary ===

Correctly Classified Instances        1009               83.3884 %
Incorrectly Classified Instances       201               16.6116 %
Kappa statistic                          0.6765
Mean absolute error                      0.1188
Root mean squared error                  0.2341
Relative absolute error                 51.7232 %
Root relative squared error             69.234  %
Total Number of Instances             1210
```

50% test: (Accuracy : 85.76%)

```
=== Summary ===

Correctly Classified Instances         741               85.7639 %
Incorrectly Classified Instances       123               14.2361 %
Kappa statistic                          0.7163
Mean absolute error                      0.1149
Root mean squared error                  0.2262
Relative absolute error                 50.0862 %
Root relative squared error             66.8308 %
Total Number of Instances              864
```

2. kNN[weka.classifiers.lazy.IBk -K 1 -W 0 -A] with default settings

10% test: (Accuracy : 86.11%)

```
=== Summary ===

Correctly Classified Instances        1339               86.1093 %
Incorrectly Classified Instances       216               13.8907 %
Kappa statistic                          0.689
Mean absolute error                      0.0754
Root mean squared error                  0.251
Relative absolute error                 33.3435 %
Root relative squared error             73.9022 %
Total Number of Instances             1555
```

30% test: (Accuracy : 87.69%)

```
=== Summary ===

Correctly Classified Instances        1061               87.686  %
Incorrectly Classified Instances       149               12.314  %
Kappa statistic                          0.735
Mean absolute error                      0.063
Root mean squared error                  0.2376
Relative absolute error                 27.4052 %
Root relative squared error             70.2621 %
Total Number of Instances             1210
```

50% test: (Accuracy : 89.46%)

```
=== Summary ===

Correctly Classified Instances         773               89.4676 %
Incorrectly Classified Instances        91               10.5324 %
Kappa statistic                          0.7732
Mean absolute error                      0.0526
Root mean squared error                  0.2145
Relative absolute error                 22.9305 %
Root relative squared error             63.3957 %
Total Number of Instances              864
```

3. SVM

10% test: (Accuracy : 78.84%)

```
=== Summary ===

Correctly Classified Instances        1226               78.8424 %
Incorrectly Classified Instances       329               21.1576 %
Kappa statistic                          0.4617
Mean absolute error                      0.2732
Root mean squared error                  0.3467
Relative absolute error                120.8616 %
Root relative squared error            102.0545 %
Total Number of Instances             1555
```

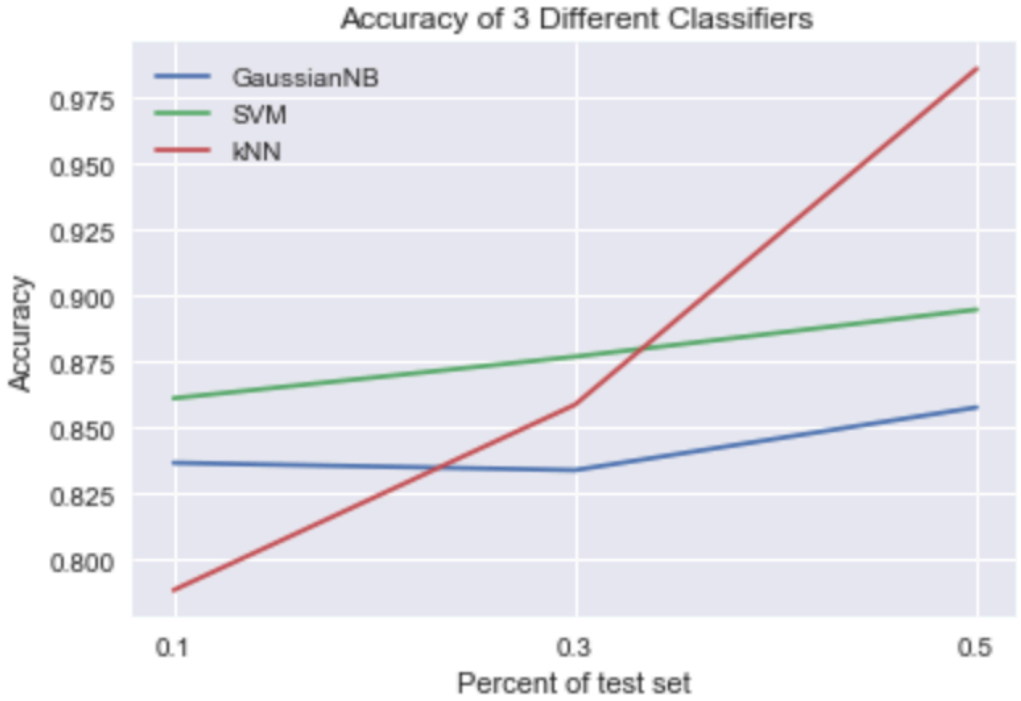
30% test: (Accuracy : 85.86%)

```
=== Summary ===

Correctly Classified Instances        1039               85.8678 %
Incorrectly Classified Instances       171               14.1322 %
Kappa statistic                          0.6902
Mean absolute error                      0.264
Root mean squared error                  0.3331
Relative absolute error                114.885  %
Root relative squared error             98.5085 %
Total Number of Instances             1210
```

50% test: (Accuracy : 85.87%)

```
=== Summary ===

Correctly Classified Instances         742               85.8796 %
Incorrectly Classified Instances       122               14.1204 %
Kappa statistic                          0.6865
Mean absolute error                      0.2643
Root mean squared error                  0.3332
Relative absolute error                115.2367 %
Root relative squared error             98.4545 %
Total Number of Instances              864
```

Visualization



It is clear that with 50% test set, kNN has the best performance, while with 10% test set, SVM has the best performance.

Section 2

In [58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn import tree
from sklearn import metrics
from sklearn.model_selection import train_test_split


from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

%matplotlib inline
```

In [59]:
```
car_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data', names = ["buying", "maint", "doors", "persons", "lug_boot", "safety", "class"])
```

In [60]:
```
car_data.info() # number of items
```

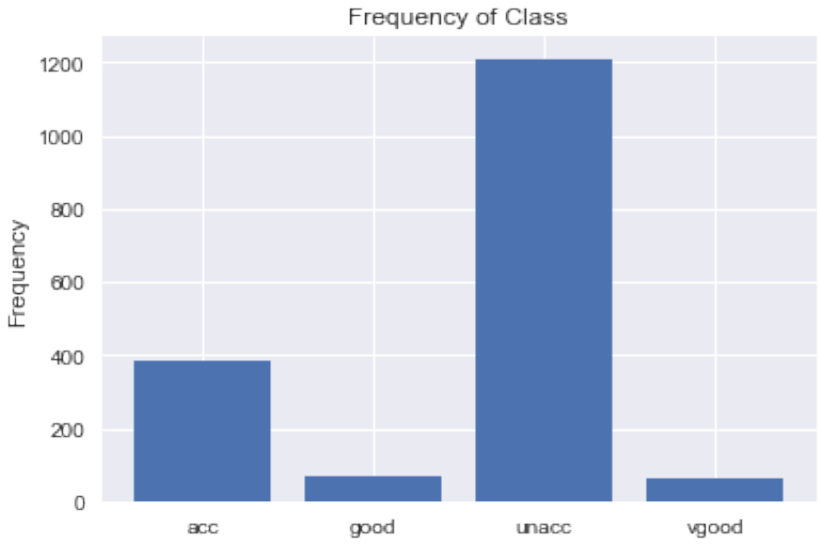
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying      1728 non-null object
maint       1728 non-null object
doors       1728 non-null object
persons     1728 non-null object
lug_boot    1728 non-null object
safety      1728 non-null object
class       1728 non-null object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [ ]:
```
# Data Set Characteristics:    Multivariate
# Number of Instances:   1728
# Attribute Characteristics: Categorical
# Number of Attributes: 6
# Associated Tasks:  Classification
# Missing Values:  No
```

In [112]:
```
class_data = car_data.groupby(['class'])
class_data.size()
```

Out[112]:
```
class
acc        384
good        69
unacc     1210
vgood       65
dtype: int64
```

In [113]:
```
names = []
count = []
for name, group in class_data:
    names.append(name)
    count.append(len(group))

y_pos = np.arange(len(names))
plt.bar(y_pos, count, align = 'center')
plt.xticks(y_pos,names)
plt.ylabel('Frequency')
plt.title('Frequency of Class')

plt.show()
```



In [ ]:
```
# The above bar chart shows the frequency of the class output
#  It shows that out of the total 1728 cars in the dataset, 384(22.28
%) were acceptable,
#  69 (4.05 %) were good, 1210 (69.85 %) were unacceptable,
#  and 65 cars (3.82%) were very good.
```

In [ ]:
```

```

In [114]:
```
# Data Cleaning :conversion of nominal attributes to numeric attribute
s
```

In [126]:
```
# Step 1: Cleaning "Buying: buying" Attribute
# 'vhigh' : 4
# 'high': 3
# 'med': 2
# 'low': 1

# Step 2: Cleaning "Maintenance: maint" Attribute
# 'vhigh' : 4
# 'high': 3
# 'med': 2
# 'low': 1

# Step 3: Cleaning 'Luggage Boot: lug_boot' Attribute
# 'Big': 3
# 'Small' : 1

# Step 4: Cleaning 'Doors: doors' Attribute
# '5more': 6
# 'more': 5

# Step 5: Cleaning 'Class: class' Attribute
# 'vgood': 4
# 'good': 3
# 'acc': 2
# 'unacc': 1
```

In [115]:
```
df = car_data.replace('vhigh', 4)
df = df.replace('high',3)
df = df.replace('med', 2)
df = df.replace('low', 1)

df = df.replace('big', 3)
df = df.replace('small', 1)

df = df.replace('5more', 6)
df = df.replace('more', 5)

df = df.replace('vgood', 4)
df = df.replace('good',3)
df = df.replace('acc', 2)
df = df.replace('unacc', 1)
```

In [ ]:
```

```

In [ ]:
```
# [TBC] Data Transformation
```

In [ ]:
```

```

In [124]:
```
# Data Split
```

In [ ]:
```
# Training and Testing % Split
# 90% : 10%
# 60% : 30%
# 50% : 50%
# 10 Folds
```

In [130]:
```
car = df.values
X,y = car[:,:6], car[:,6]
X,y = X.astype(int), y.astype(int)
```

In [ ]:
```

```

In [ ]:
```

```

In [163]:
```
# Classification: 1. Naive Bayesian 2. kNN 3. Support Vector Machines
```

In [216]:
```
models = {}
```

In [215]:
```
def addModel(models, name, acc):
    if name not in models:
        models[name] = [acc]
    else:
        models[name].append(acc)
```

In [ ]:
```
# 1. Naive Bayes
# Naive Bayes uses Bayes Theorem to model the conditional relationship
of
# each attribute to the class variable.
```

In [197]:
```
def GNB(X_train, y_train, X_test,y_test):
    # fit a Naive Bayes model to the data
    model = GaussianNB()
    model.fit(X_train, y_train)
    print(model)

    acc = predict(model, X_test, y_test)

    name = 'GaussianNB'
    addModel(models, name, acc)
```

In [ ]:
```
# 2. kNN
```

In [198]:
```
def kNN(X_train, y_train, X_test,y_test):
    # fit a Naive Bayes model to the data
    model = KNeighborsClassifier()
    model.fit(X_train, y_train)
    print(model)

    acc = predict(model, X_test, y_test)

    name = 'kNN'
    addModel(models, name, acc)
```

In [ ]:
```
# 3. Support Vector Machines (SVM)
# Support Vector Machines (SVM) are a method that uses points in a tra
nsformed problem space
# that best separate classes into two groups.
```

In [199]:
```
def svm(X_train, y_train, X_test,y_test):
    # fit a SVM model to the data
    model = SVC()
    model.fit(X_train, y_train)
    print(model)

    acc = predict(model, X_test, y_test)

    name = 'SVM'
    addModel(models, name, acc)
```

In [ ]:

In [187]:
```
def split(test_percent):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=
test_percent,random_state=0)
    GNB(X_train,  y_train, X_test,y_test)
    kNN(X_train,  y_train, X_test,y_test)
    svm(X_train,  y_train, X_test,y_test)
```

In [ ]:
```

```

In [219]:
```
percent = [0.1, 0.3, 0.5]
```

In [217]:
```
for p in percent:
    split(p)
```

```
GaussianNB(priors=None)
             precision    recall  f1-score   support

          1       0.89      0.88      0.88       120
          2       0.40      0.19      0.26        42
          3       0.33      0.29      0.31         7
          4       0.14      1.00      0.24         4

avg / total       0.73      0.69      0.69       173

Confusion Matrix:
[[105  12   0   3]
 [ 12   8   4  18]
 [  1   0   2   4]
 [  0   0   0   4]]

Accuracy: 0.751445086705
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
             precision    recall  f1-score   support

          1       0.96      0.97      0.97       120
          2       0.89      0.81      0.85        42
          3       0.67      0.86      0.75         7
          4       0.75      0.75      0.75         4

avg / total       0.93      0.92      0.92       173

Confusion Matrix:
[[117   3   0   0]
 [  5  34   3   0]
```

```
 [  0   0   6   1]
 [  0   1   0   3]]

Accuracy: 0.963391136802
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf'
,
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
             precision    recall  f1-score   support

          1       0.99      0.95      0.97       120
          2       0.81      0.90      0.85        42
          3       0.50      0.43      0.46         7
          4       0.60      0.75      0.67         4

avg / total       0.92      0.91      0.91       173

Confusion Matrix:
[[114   6   0   0]
 [  1  38   3   0]
 [  0   2   3   2]
 [  0   1   0   3]]

Accuracy: 0.961464354528
GaussianNB(priors=None)
             precision    recall  f1-score   support

          1       0.93      0.92      0.93       363
          2       0.58      0.34      0.43       115
          3       0.38      0.20      0.26        25
          4       0.20      1.00      0.33        16

avg / total       0.80      0.76      0.77       519

Confusion Matrix:
[[334  21   0   8]
 [ 23  39   8  45]
 [  2   7   5  11]
 [  0   0   0  16]]

Accuracy: 0.7591522158
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
             precision    recall  f1-score   support

          1       0.97      0.98      0.97       363
          2       0.86      0.90      0.88       115
```

```
          3       0.85      0.68      0.76        25
          4       0.91      0.62      0.74        16

avg / total       0.94      0.94      0.93       519

Confusion Matrix:
[[355   8   0   0]
 [  9 104   2   0]
 [  3   4  17   1]
 [  0   5   1  10]]

Accuracy: 0.936416184971
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf'
,
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
             precision    recall  f1-score   support

          1       0.99      0.98      0.98       363
          2       0.88      0.93      0.90       115
          3       0.86      0.76      0.81        25
          4       0.88      0.88      0.88        16

avg / total       0.95      0.95      0.95       519

Confusion Matrix:
[[354   9   0   0]
 [  5 107   3   0]
 [  0   4  19   2]
 [  0   2   0  14]]

Accuracy: 0.95183044316
GaussianNB(priors=None)
             precision    recall  f1-score   support

          1       0.91      0.93      0.92       599
          2       0.63      0.32      0.43       193
          3       0.42      0.25      0.31        32
          4       0.30      1.00      0.46        40

avg / total       0.80      0.77      0.77       864

Confusion Matrix:
[[559  27   1  12]
 [ 53  62  10  68]
 [  1   9   8  14]
 [  0   0   0  40]]

Accuracy: 0.757225433526
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
             precision    recall  f1-score   support

          1       0.97      0.97      0.97       599
          2       0.80      0.88      0.84       193
          3       0.59      0.62      0.61        32
          4       1.00      0.42      0.60        40

avg / total       0.92      0.91      0.91       864

Confusion Matrix:
[[582  17   0   0]
 [ 20 169   4   0]
 [  1  11  20   0]
 [  0  13  10  17]]

Accuracy: 0.915221579961
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf'
,
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
             precision    recall  f1-score   support

          1       0.97      0.97      0.97       599
          2       0.82      0.88      0.84       193
          3       0.68      0.81      0.74        32
          4       1.00      0.53      0.69        40

avg / total       0.93      0.92      0.92       864

Confusion Matrix:
[[579  19   1   0]
 [ 19 169   5   0]
 [  0   6  26   0]
 [  0  13   6  21]]

Accuracy: 0.934489402697
```

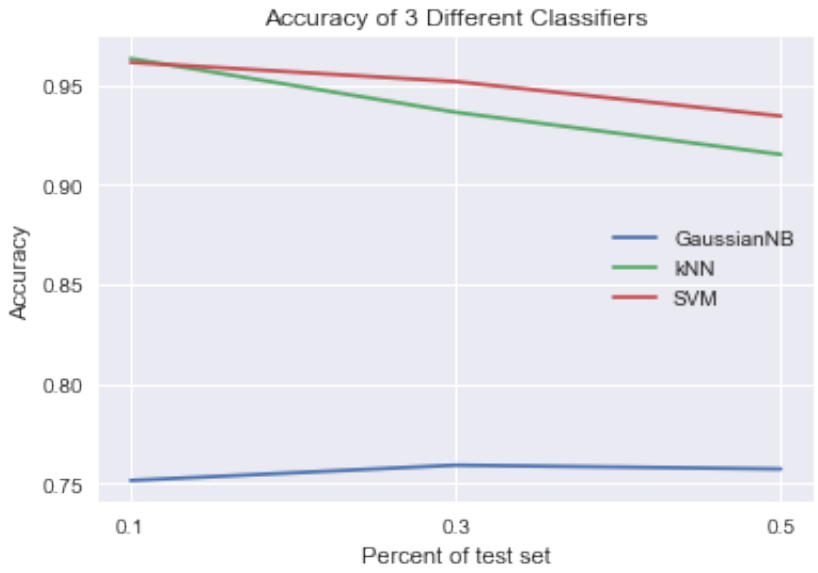
In [ ]:
```
colors = list("rgbcmyk")

for data_dict in models.values():
   x = data_dict.keys()
   y = data_dict.values()
   plt.scatter(x,y,color=colors.pop())

plt.legend(d.keys())
plt.show()
```

In [218]:
```
models
```

Out[218]:
```
{'GaussianNB': [0.75144508670520227, 0.75915221579961467, 0.75722543
352601157],
 'SVM': [0.96146435452793833, 0.95183044315992293, 0.934489402697495
14],
 'kNN': [0.96339113680154143, 0.93641618497109824, 0.915221579961464
36]}
```

In [230]:
```
for key, data in models.items():
    plt.plot(percent, data, label = key)

plt.xlabel("Percent of test set")
plt.ylabel('Accuracy')
plt.title('Accuracy of 3 Different Classifiers')
plt.xticks(percent)
plt.legend()
```

Out[230]:
```
<matplotlib.legend.Legend at 0x10fe6ec18>
```



In [ ]:
```

```