# Python_HW02

March 15, 2021

## 1 Homework: Performance Assessment

### 1.1 Use both R and Python to answer the following questions:

1. Using the census data set, choose a few meaningful categorical features as predictors and Income astarget.
2. Create train- and test data using a fixed split (use 1/3 for test set).
3. Fit a k-NN-model and a naive Bayes model. Tune k-NN using 10-times CV.
4. Predict the performances on the test set. Create the confusion matrices and compare the two classifiers in terms of Accuracy, Recall and Precision.
5. Create an ROC-curve for the naive Bayes model. Choose a good threshold, create new predictions using this threshold on the test set and again create the performance measures. Compare with the previous results (default threshold).

## 2 0 Data Preperation

### 2.1 1 Using the census data set, choose a few meaningful categorical features as predictors and Income astarget.

### 2.2 2 Create train- and test data using a fixed split (use 1/3 for test set).

[253]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import CategoricalNB
from sklearn import preprocessing
from sklearn import metrics
```

[254]:
```
census = pd.read_csv("census.csv")

census
```

[254]:

| | age | workclass | fnlwgt | education | education.num | \ |
|---|---|---|---|---|---|---|
| 0 | 39 | State-gov | 77516 | Bachelors | 13 | |
| 1 | 50 | Self-emp-not-inc | 83311 | Bachelors | 13 | |
| 2 | 38 | Private | 215646 | HS-grad | 9 | |
| 3 | 53 | Private | 234721 | 11th | 7 | |
| 4 | 28 | Private | 338409 | Bachelors | 13 | |

```
…    …                …       …          …              …
32556   27           Private  257302  Assoc-acdm             12
32557   40           Private  154374     HS-grad              9
32558   58           Private  151910     HS-grad              9
32559   22           Private  201490     HS-grad              9
32560   52      Self-emp-inc  287927     HS-grad              9

           marital.status         occupation   relationship   race     sex  \
0           Never-married       Adm-clerical  Not-in-family  White    Male
1      Married-civ-spouse    Exec-managerial        Husband  White    Male
2                Divorced  Handlers-cleaners  Not-in-family  White    Male
3      Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4      Married-civ-spouse     Prof-specialty           Wife  Black  Female
…                     …                  …              …    …      …
32556  Married-civ-spouse       Tech-support           Wife  White  Female
32557  Married-civ-spouse  Machine-op-inspct        Husband  White    Male
32558             Widowed       Adm-clerical      Unmarried  White  Female
32559       Never-married       Adm-clerical      Own-child  White    Male
32560  Married-civ-spouse    Exec-managerial           Wife  White  Female

       capital.gain  capital.loss  hours.per.week native.country income
0              2174             0              40  United-States  <=50K
1                 0             0              13  United-States  <=50K
2                 0             0              40  United-States  <=50K
3                 0             0              40  United-States  <=50K
4                 0             0              40           Cuba  <=50K
…               …             …               …              …    …
32556             0             0              38  United-States  <=50K
32557             0             0              40  United-States   >50K
32558             0             0              40  United-States  <=50K
32559             0             0              20  United-States  <=50K
32560         15024             0              40  United-States   >50K

[32561 rows x 15 columns]
```

[256]:
```
census['income'] = census['income'].replace("<=50K",0).replace(">50K",1)

census['income'] = census['income'].astype('category')
census['income'].cat.categories
```

[256]:
```
Int64Index([0, 1], dtype='int64')
```

[257]:
```
nb = census[["workclass","education","marital.
 ↪status","occupation","sex","income"]]

for col in ["workclass","education","marital.
 ↪status","occupation","sex","income"]:
```

```
    nb[col]=nb[col].astype('category')

nb
```

```
<ipython-input-257-1b7a11ef57df>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  nb[col]=nb[col].astype('category')
```

[257]:

| | workclass | education | marital.status | occupation | sex | income |
|---|---|---|---|---|---|---|
| 0 | State-gov | Bachelors | Never-married | Adm-clerical | Male | 0 |
| 1 | Self-emp-not-inc | Bachelors | Married-civ-spouse | Exec-managerial | Male | 0 |
| 2 | Private | HS-grad | Divorced | Handlers-cleaners | Male | 0 |
| 3 | Private | 11th | Married-civ-spouse | Handlers-cleaners | Male | 0 |
| 4 | Private | Bachelors | Married-civ-spouse | Prof-specialty | Female | 0 |
| … | … | … | … | … | … | … |
| 32556 | Private | Assoc-acdm | Married-civ-spouse | Tech-support | Female | 0 |
| 32557 | Private | HS-grad | Married-civ-spouse | Machine-op-inspct | Male | 1 |
| 32558 | Private | HS-grad | Widowed | Adm-clerical | Female | 0 |
| 32559 | Private | HS-grad | Never-married | Adm-clerical | Male | 0 |
| 32560 | Self-emp-inc | HS-grad | Married-civ-spouse | Exec-managerial | Female | 1 |

[32561 rows x 6 columns]

[258]:
```
train,test =train, test = train_test_split(nb, test_size=0.3)
train.reset_index(drop=True, inplace=True)
test.reset_index(drop=True, inplace=True)

## One-Hot Encoding

enc=np.array(["workclass","education","marital.status","occupation","sex"])
```

```
train_predictors= pd.get_dummies(train.drop("income",axis=1),columns=enc)
test_predictors= pd.get_dummies(test.drop("income",axis=1),columns=enc)
```

# 3 3. Fit a k-NN-model and a naive Bayes model. Tune k-NN using 10-times CV.

## 3.1 NB

[259]:
```
model = CategoricalNB(alpha=1)
model.fit(train_predictors,train["income"])

prediction= model.predict(test_predictors)
```

## 3.2 KNN

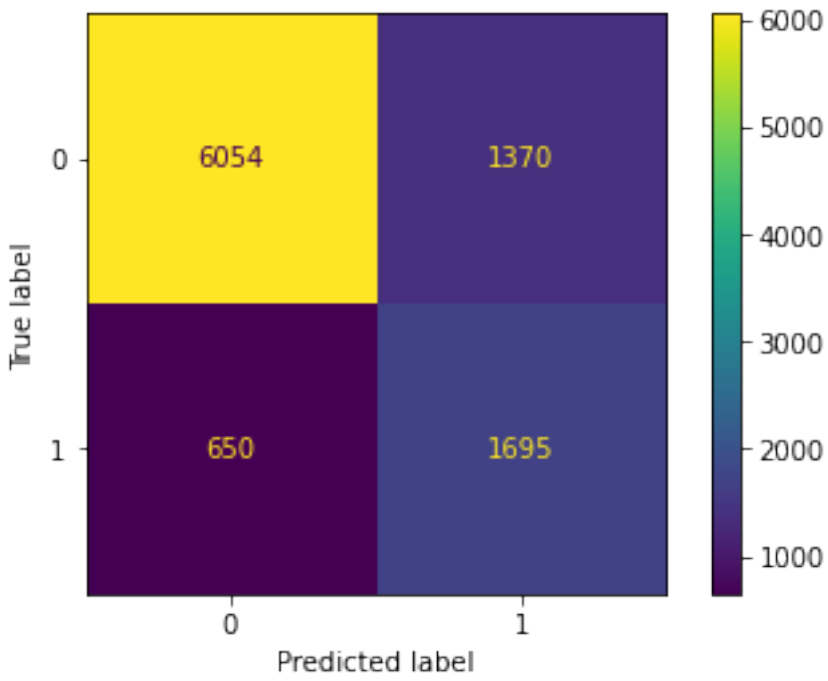
[245]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
knn_data= census[["age","education.num","capital.gain","capital.loss","hours.
 ↪per.week","income"]]

knn_train,knn_test =train, test = train_test_split(knn_data, test_size=0.3)

model = KNeighborsClassifier()


# with GridSearch
KSearch = GridSearchCV(
    estimator= model,
    param_grid={'n_neighbors' : list(range(1,10))},
    scoring = 'accuracy',
    cv = 10
)


search=KSearch.fit(knn_train.drop("income",axis=1), knn_train["income"])
result= search.best_estimator_


print('Best k : %d' % result.get_params()['n_neighbors'])
```

```
Best k : 8
```

## 4 4. Predict the performances on the test set. Create the confusion matrices and compare the two classifiers in terms of Accuracy, Recall and Precision.

[260]:
```
metrics.plot_confusion_matrix(model,test_predictors, test["income"].to_numpy())
```

[260]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28dcf3b82e0>
```

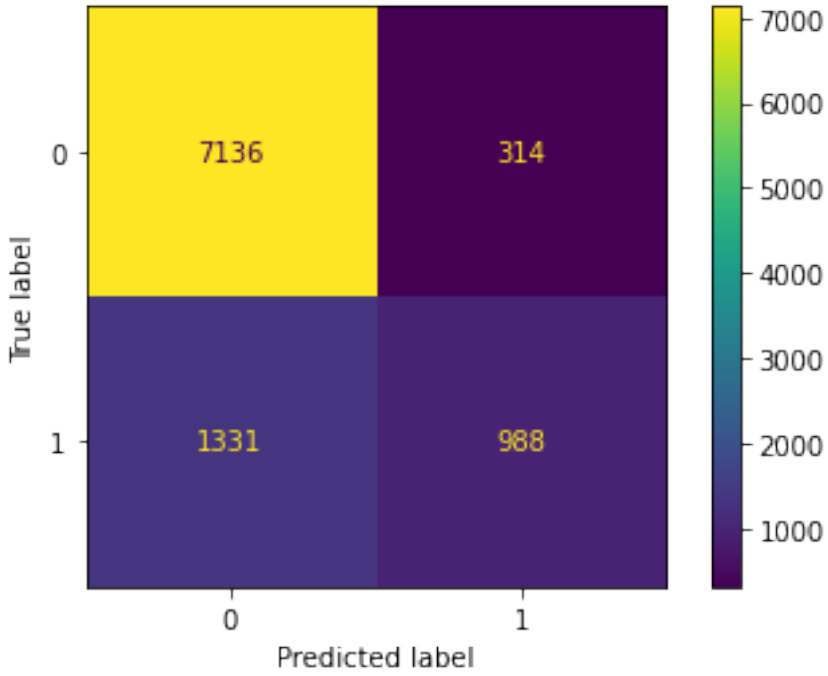



[262]:
```
print(metrics.classification_report(test["income"],␣
↪prediction,target_names=['<50k','>=50k']))
```

```
              precision    recall  f1-score   support

       <50k       0.90      0.82      0.86      7424
      >=50k       0.55      0.72      0.63      2345

    accuracy                           0.79      9769
   macro avg       0.73      0.77      0.74      9769
weighted avg       0.82      0.79      0.80      9769
```

[265]:
```
metrics.plot_confusion_matrix(search,knn_test.
↪drop("income",axis=1),knn_test["income"])
```

[265]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28df5515d00>




[266]:
```
knn_pred= search.predict(knn_test.drop("income",axis=1))
print(metrics.classification_report(knn_test["income"],␣
 ↪knn_pred,target_names=['<50k','>=50k']))
```

```
              precision    recall  f1-score   support

       <50k       0.84      0.96      0.90      7450
      >=50k       0.76      0.43      0.55      2319

    accuracy                           0.83      9769
   macro avg       0.80      0.69      0.72      9769
weighted avg       0.82      0.83      0.81      9769
```