CS412 - Machine Learning - 2020

Homework 2

100 pts

Goal

The goal of this homework is to get familiar feature handling and cross validation.

Dataset

German Credit Risk dataset, prepared by Prof. Hoffman, classifies each person as having a good or bad credit risk. The dataset that we use consists of both numerical and categorical features.

Task

Build a k-NN classifier with scikit-learn library to classify people as bad or good risks for the german credit dataset.

Software

Documentation for the necessary functions can be accessed from the link below.

http://scikit-learn.org/stable/supervised_learning.html

Submission

Follow the instructions at the end.

1) Initialize

First, make a copy of this notebook in your drive

```
# Mount to your drive, in this way you can reach files that are in your drive
# Run this cell
# Go through the link that will be showed below
# Select your google drive account and copy authorization code and paste here in ou
# You can also follow the steps from that link
# https://medium.com/ml-book/simplest-way-to-open-files-from-google-drive-in-google

```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

2) Load Dataset

To start working for your homework, take a copy of the folder, given in the below link to your own google drive. You find the train and test data under this folder.

https://drive.google.com/drive/folders/1DbW6VxLKZv2oqFn9SwxAnVadmn1_nPXi?usp=sharing

After copy the folder, copy the path of the train and test dataset to paste them in the below cell to load your data.

```
import pandas as pd

train_df = pd.read_csv('/content/drive/My Drive/german_credit_test.csv')
test_df = pd.read_csv('/content/drive/My Drive/german_credit_test.csv')
```

3) Optional - Analyze the Dataset

You can use the functions of the pandas library to analyze your train dataset in detail - **this part is OPTIONAL - look around the data as you wish**.

- Display the number of instances and features in the train ***(shape function can be used)**
- Display 5 random examples from the train ***(sample function can be used)**
- Display the information about each features ***(info method can be used)**

```
# Print shape
print("Train data dimensionality: ", train_df.shape )
```



```
print("Examples from train data: ")
train_df.head()
```

```
Train data dimensionality:  (200, 13)
Examples from train data:
```

	AccountStatus	Duration	CreditHistory	CreditAmount	SavingsAccount	Employme
0	A12	30	A31	3496	A64	

```
1 # Print the information about the dataset
2 print("Information about train data ")
3 train_df.info()
4
5 print("Information about test data ")
6 test_df.info()
```

```
Information about train data
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   AccountStatus      200 non-null    object
 1   Duration           200 non-null    int64
 2   CreditHistory      200 non-null    object
 3   CreditAmount       200 non-null    int64
 4   SavingsAccount     200 non-null    object
 5   EmploymentSince    200 non-null    object
 6   PercentOfIncome    200 non-null    int64
 7   PersonalStatus     200 non-null    object
 8   Property           200 non-null    object
 9   Age                200 non-null    int64
 10  OtherInstallPlans  200 non-null    object
 11  Housing            180 non-null    object
 12  Risk               200 non-null    int64
dtypes: int64(5), object(8)
memory usage: 20.4+ KB
Information about test data
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```



```
 3   CreditAmount       200 non-null    int64
 4   SavingsAccount     200 non-null    object
 5   EmploymentSince    200 non-null    object
 6   PercentOfIncome    200 non-null    int64
 7   PersonalStatus     200 non-null    object
 8   Property           200 non-null    object
 9   Age                200 non-null    int64
 10  OtherInstallPlans  200 non-null    object
 11  Housing            180 non-null    object
 12  Risk               200 non-null    int64
dtypes: int64(5), object(8)
memory usage: 20.4+ KB
```

4) Define your train and test labels

- Define labels for both train and test data in new arrays
- And remove the label column from both train and test sets do tht it is not used as a feature!

(**you can use pop method**)

```
# Define labels
train_label = train_df.pop('Risk')
test_label = test_df.pop('Risk')
```

5) Handle missing values if any

- Print the columns that have **NaN** values (**isnull** method can be used)
- You can impute missing values with mode of that feature or remove samples or attributes
- To impute the test set, you should use the mode values that you obtain from **train** set, as **you should not be looking at your test data to gain any information or advantage.**

```
# Print columns with NaN values
print("NaN value percentage in Housing: %", train_df['Housing'].isnull().sum() / le

```

```
NaN value percentage in Housing: % 10.0
```

```
# Since 10% is a reasonable and the data is limited, we need to impute missing valu
# Impute missing values by replacing with train set mode value

train_df['Housing'] = train_df['Housing'].fillna(train_df['Housing'].mode()[0])

#mode values of train data is used since we dont have information about test set
```


6) Transform categorical / ordinal features

- Transform all categorical / ordinal features using the methods that you have learnt in lectures and recitation 4 for both train and test data
- You saw the dictionary use for mapping in recitation. (You can use **replace function** to assign new values to the categories of a column).
- The class of the categorical attributes in the dataset are defined as follows:

- Status of existing checking account
  - A11 : ... < 0 DM
  - A12 : 0 <= ... < 200 DM
  - A13 : ... >= 200 DM / salary assignments for at least 1 year
  - A14 : no checking account
- Credit history
  - A30 : no credits taken/all credits paid back duly
  - A31 : all credits at this bank paid back duly
  - A32 : existing credits paid back duly till now
  - A33 : delay in paying off in the past
  - A34 : critical account/other credits existing (not at this bank)
- Savings account
  - A61 : ... < 100 DM
  - A62 : 100 <= ... < 500 DM
  - A63 : 500 <= ... < 1000 DM
  - A64 : .. >= 1000 DM
  - A65 : unknown/ no savings account
- Employment Since
  - A71 : unemployed
  - A72 : ... < 1 year
  - A73 : 1 <= ... < 4 years
  - A74 : 4 <= ... < 7 years
  - A75 : .. >= 7 years
- Personal Status
  - A91 : male : divorced/separated

  - A94 : male : married/widowed
  - A95 : female : single
- Property
  - A121 : real estate
  - A122 : if not A121 : building society savings agreement/life insurance
  - A123 : if not A121/A122 : car or other, not in attribute 6
  - A124 : unknown / no property
- OtherInstallPlans

- A141 : bank
- A142 : stores
- A143 : none

- Housing
  - A151 : rent
  - A152 : own
  - A153 : for free

```
# Unique values in each non-numerical feature
for col in train_df.select_dtypes('object'):
  print(col, train_df[col].unique())
```

```
AccountStatus ['A12' 'A11' 'A13' 'A14']
CreditHistory ['A31' 'A34' 'A32' 'A30' 'A33']
SavingsAccount ['A64' 'A61' 'A65' 'A63' 'A62']
EmploymentSince ['A73' 'A71' 'A72' 'A75' 'A74']
PersonalStatus ['A93' 'A92' 'A91' 'A94']
Property ['A123' 'A121' 'A124' 'A122']
OtherInstallPlans ['A142' 'A143' 'A141']
Housing ['A152' 'A151' 'A153']
```

```
# Transform the categorical / ordinal attributes

#Transforming ordinal attributes
#AccountStatus, SavingsAccount and EmploymentSince columns

print(train_df.columns)

account_map={'A11':0, 'A12':1, 'A13':2, 'A14':3}
train_df["AccountStatus"] = train_df["AccountStatus"].replace(account_map)
test_df['AccountStatus'] = test_df['AccountStatus'].replace(account_map)

savings_map={'A61':0, 'A62':1, 'A63':2, 'A64':3, 'A65':4}
train_df['SavingsAccount'] = train_df['SavingsAccount'].replace(savings_map)
```



```
employment_map={'A71':0, 'A72':1, 'A73':2, 'A74':3, 'A75':4}
train_df['EmploymentSince'] = train_df['EmploymentSince'].replace(employment_map)
test_df['EmploymentSince'] = test_df['EmploymentSince'].replace(employment_map)


```

```
Index(['AccountStatus', 'Duration', 'CreditHistory', 'CreditAmount',
       'SavingsAccount', 'EmploymentSince', 'PercentOfIncome',
       'PersonalStatus', 'Property', 'Age', 'OtherInstallPlans', 'Housing'],
      dtype='object')
```

```
#Dummy columns
```

```
personal_status_dummies=pd.get_dummies(train_df['PersonalStatus'], prefix='personal
personal_status_dummies.head()
```

	personalStatus_A91	personalStatus_A92	personalStatus_A93	personalStatus_A9
0	0	0	1	0
1	0	1	0	0
2	1	0	0	0
3	0	0	1	0
4	0	1	0	0

```
#Dummy columns
other_dummies=pd.get_dummies(train_df['OtherInstallPlans'], prefix='otherInstallPla
other_dummies.head()
```

	otherInstallPlans_A141	otherInstallPlans_A142	otherInstallPlans_A143
0	0	1	0
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1

```
train_df.pop('PersonalStatus')
test_df.pop('PersonalStatus')

train_df.pop('OtherInstallPlans')
test_df.pop('OtherInstallPlans')
```

```
0      A142
1      A143
4      A143
       ...
195    A143
196    A143
197    A143
198    A143
199    A143
Name: OtherInstallPlans, Length: 200, dtype: object
```



```
#Transforming categorical variables

from sklearn.preprocessing import OneHotEncoder

```

```
enc= OneHotEncoder(handle_unknown='ignore')
dummies = enc.fit_transform(train_df[['CreditHistory', 'Property', 'Housing']]).toa
dummies=pd.DataFrame(dummies)

train_df = pd.merge(train_df, dummies, right_index=True, left_index=True)
train_df=train_df.drop(columns=['CreditHistory', 'Property', 'Housing'])
```

```
dummies_test=enc.transform(test_df[['CreditHistory', 'Property', 'Housing']]).toarı
dummies_test=pd.DataFrame(dummies_test)

test_df = pd.merge(test_df, dummies_test, right_index=True, left_index=True)
test_df=test_df.drop(columns=['CreditHistory', 'Property', 'Housing'])
```

7) Build a k-NN classifier on training data and perform models selection using 5 fold cross validation

- Initialize k-NN classifiers with **k= 5, 10, 15**
- Calculate the cross validation scores using cross_al_score method, number of folds is 5.
- Note: Xval is performed on training data! Do not use test data in any way and do not separate a hold-out validation set, rather use cross-validation.

Documentation of the cross_val_score method:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score

- Stores the average accuracies of these folds
- Select the value of k using the cross validation results.


```
from statistics import mean

# k values
kVals = [5,10,15]

# Save the accuracies of each value of kVal in [accuracies] variable
accuracies = []

# Loop over values of k for the k-Nearest Neighbor classifier
for k in kVals:
  # Initialize a k-NN classifier with k neighbors
  classifier= KNeighborsClassifier(n_neighbors=k)
```

```
  # Calculate the 5 fold cross validation scores using cross_val_score
  # cv parameter: number of folds, in our case it must be 5
  scores = cross_val_score(classifier, train_df, train_label, cv=5)

  # Stores the average accuracies of the scores in accuracies variable, you can use
  accuracies.append(mean(scores))

print(accuracies)
```

```
[0.575, 0.635, 0.625]
```

8) Retrain using all training data and test on test set

- Train a classifier with the chosen k value of the best classifier using **all training data**.

Note: k-NN training involves no explicit training, but this is what we would do after model selection with decision trees or any other ML approach (we had 5 diff. models -one for each fold - for each k in the previous step - dont know which one to submit. Even if we picked the best one, it does not use all training samples.

- Predict the labels of testing data
- Report the accuracy

```
from sklearn.metrics import accuracy_score
import numpy as np

# Train the best classifier using all training set
best_classifier= KNeighborsClassifier(n_neighbors=kVals[np.argmax(accuracies)])
best_classifier.fit(train_df, train_label)

# Estimate the prediction of the test data
preds=best_classifier.predict(test_df)
accuracy = accuracy_score(test_label, preds)
print(accuracy)

```



```
0.705
```

9) Bonus (5pts)

There is a limited bonus for any extra work that you may use and improve the above results.

You may try a larger k values, scale input features, remove some features, ... Please **do not overdo**, maybe spend another 30-60min on this. The idea is not do an exhaustive search (which wont help your understanding of ML process), but just to give some extra points to those who may look at the problem a little more comprehensively.

If you obtain better results than the above, please indicate the best model you have found and the corresponding accuracy.

E.g. using feature normalization ..... and removing .... features and using a value k=...., I have obtained ... % accuracy

1

10) Notebook & Report

The objective of this project is to solve the problem of classifying customers as bad and good risks by using German Credit Dataset.

There were 10% of missing values in the Housing column. The missing values of the Housing column of train and test sets are imputed by replacing with train set 'Housing' mode. The ordinal attributes of 'AccountStatus', 'SavingsAccount' and 'EmploymentSince' columns, are replaced with integers. Moreover in terms of transforming categorical variables are transformed into numeric array.

The average validation accuracies are 0.575, 0.635, 0.625 respectively to the k values 5,10,15. Since the accuracy is highest when k is 10, k=10 is selected as the best knn classifier.

We have obtained the best result with the best classifier (parameters: k=10, cv=5), giving classification accuracy of 0.705 on test data.

11) Submission

on the link and go there and run (and possibly also modify) your code.

For us to be able to modify, in case of errors etc, **you should get your "share link" as **share with anyone in edit mode**

Also submit your notebook as pdf as attachment, choose print and save as PDF, save with hw2-lastname-firstname.pdf to facilitate grading.

Otomatik kaydetme gerçekleştirilemedi. Bu dosya uzaktan veya başka bir sekmede güncellenmiş. Farkı göster