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 output and p
# You can also follow the steps from that link
# https://medium.com/ml-book/simplest-way-to-open-files-from-google-drive-in-google-colab-fae
```

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

```
     Mounted at /content/drive
```

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/Copy of german_credit_train.csv')
test_df = pd.read_csv('/content/drive/My Drive/Copy of 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)**

```
import numpy as np
# Print shape
print("Train data dimensionality: ", )
print(train_df.shape)

# Print random 5 rows
print("Examples from train data: ")
train_df.head()
#train_df.sample()
```

```
Train data dimensionality:
(800, 13)
Examples from train data:
```

	AccountStatus	Duration	CreditHistory	CreditAmount	SavingsAccount	EmploymentSinc
0	A14	12	A32	2859	A65	A7
1	A11	9	A32	2136	A61	A7

```
# Print the information about the dataset
print("Information about train data ", )
train_df.info()
```

```
Information about train data
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   AccountStatus     800 non-null    object
 1   Duration          800 non-null    int64
 2   CreditHistory     800 non-null    object
 3   CreditAmount      800 non-null    int64
 4   SavingsAccount    800 non-null    object
 5   EmploymentSince   800 non-null    object
 6   PercentOfIncome   800 non-null    int64
 7   PersonalStatus    800 non-null    object
 8   Property          800 non-null    object
 9   Age               800 non-null    int64
 10  OtherInstallPlans 800 non-null    object
 11  Housing           720 non-null    object
 12  Risk              800 non-null    int64
dtypes: int64(5), object(8)
memory usage: 81.4+ 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")

#train_df.head()
```

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("Number of missing values in the columns of train set:")
print(train_df.isnull().sum())
print("\n")
print("Number of missing values in the columns of test set:")
print(test_df.isnull().sum())
```

```
Number of missing values in the columns of train set:
AccountStatus         0
Duration              0
CreditHistory         0
CreditAmount          0
SavingsAccount        0
EmploymentSince       0
PercentOfIncome       0
PersonalStatus        0
Property              0
Age                   0
OtherInstallPlans     0
Housing              80
dtype: int64


Number of missing values in the columns of test set:
AccountStatus         0
Duration              0
CreditHistory         0
CreditAmount          0
SavingsAccount        0
EmploymentSince       0
PercentOfIncome       0
PersonalStatus        0
Property              0
Age                   0
OtherInstallPlans     0
Housing              20
dtype: int64
```

```
# Impute missing values by replacing with mode value
train_df['Housing'] = train_df['Housing'].fillna(train_df['Housing'].mode()[0])
test_df['Housing'] = test_df['Housing'].fillna(train_df['Housing'].mode()[0])

#print(train_df.isnull().sum())
#print(test_df.isnull().sum())
```

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
        - A92 : female : divorced/separated/married
        - A93 : male : single
        - 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

```
# Transform the categorical / ordinal attributes
from sklearn.preprocessing import OneHotEncoder

#using mapping for ordinal features
#Account Status, Savings Account and Employment Since taken as ordinal features
account_mapping = {'A11':1,'A12':2,'A13':3,'A14':0}
train_df["AccountStatus"] = train_df["AccountStatus"].replace(account_mapping)
test_df["AccountStatus"] = test_df["AccountStatus"].replace(account_mapping)

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

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

#using One Hot Encoder for categorical features
#Credit History, Personal Status, Housing, Property and Other Install Plans taken as categori
enc = OneHotEncoder(handle_unknown='ignore')
dummies_train = enc.fit_transform(train_df[['CreditHistory', 'PersonalStatus', 'Housing',
                                            'Property','OtherInstallPlans']]).toarray() #for
dummies_train = pd.DataFrame(dummies_train)
train_df = pd.merge(train_df,dummies_train,right_index=True,left_index=True)
train_df = train_df.drop(columns=['CreditHistory', 'PersonalStatus', 'Housing',
                                  'Property', 'OtherInstallPlans']) #dropping the old columns

dummies_test = enc.fit_transform(test_df[['CreditHistory', 'PersonalStatus', 'Housing',
                                          'Property','OtherInstallPlans']]).toarray() #for te
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', 'PersonalStatus', 'Housing',
                                'Property', 'OtherInstallPlans']) #dropping the old columns
#train_df.head()
```

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 sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from statistics import mean
from statistics import stdev #to calculate standar deviation between folds
# 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
  knn = 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(knn,train_df,train_label, cv=5)
  #print(scores)
  stdevfolds = stdev(scores) #calculated for the report part, scores taken as samples
  print("When k =", k, "standard deviation between the folds is:", stdevfolds)

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

for i in range(len(kVals)):
  print("When k =", kVals[i], "accuracy score is", accuracies[i])
```

```
When k = 5 standard deviation between the folds is: 0.036816691187557836
When k = 10 standard deviation between the folds is: 0.006846531968814583
When k = 15 standard deviation between the folds is: 0.012960275845829835
When k = 5 accuracy score is 0.67625
When k = 10 accuracy score is 0.7075
When k = 15 accuracy score is 0.71
```

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

# Train the best classifier using all training set
print("The best classifier is k =", kVals[np.argmax(accuracies)])
best_knn = KNeighborsClassifier(n_neighbors = kVals[np.argmax(accuracies)])
best_knn.fit(train_df,train_label)

# Estimate the prediction of the test data
test_prediction = best_knn.predict(test_df)

# Print accuracy of test data
acc = accuracy_score(test_label,test_prediction)
print("The accuracy on test set is", acc)
```

```
The best classifier is k = 15
The accuracy on test set is 0.665
```

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

```
#Trying with removing Credit Amount and k=25
train_df_without_credit = train_df.drop(columns=["CreditAmount"])
test_df_without_credit = test_df.drop(columns=["CreditAmount"])

best_knn = KNeighborsClassifier(n_neighbors = 25)
best_knn.fit(train_df_without_credit,train_label)

prediction = best_knn.predict(test_df_without_credit)
print("Removing Credit Amount and using a value k = 25, I have obtained", accuracy_score(test
```

```
     Removing Credit Amount and using a value k = 25, I have obtained 0.69 accuracy.
```

```
#Trying with normalized features
train_df_normalized = (train_df - train_df.min())/(train_df.max()-train_df.min())
test_df_normalized = (test_df - test_df.min())/(test_df.max()-test_df.min())

best_knn = KNeighborsClassifier(n_neighbors = 15)
best_knn.fit(train_df_normalized,train_label)

prediction = best_knn.predict(test_df_normalized)
print("Normalizing all the features and using a k value = 15, I have obtained", accuracy_scor
print("This is the best model I have found.")
```

```
     Normalizing all the features and using a k value = 15, I have obtained 0.705 accuracy.
     This is the best model I have found.
```

10) Notebook & Report

Notebook: We may just look at your notebook results; so make sure each cell is run and outputs are there.

Report: Write an at most 1/2 page summary of your approach to this problem at the end of your notebook; this should be like an abstract of a paper or the executive summary.

Must include statements such as:

(Include the problem definition: 1-2 lines)

(Talk about any preprocessing you do, How you handle missing values and categorical features)

(Give the average validation accuracies for different k values and standard deviations between 5 folds of each k values, state which one you selected)

(State what your test results are with the chosen method, parameters: e.g. "We have obtained the best results with the ….. classifier (parameters=....) , giving classification accuracy of …% on test data…."

State if there is any **bonus** work...

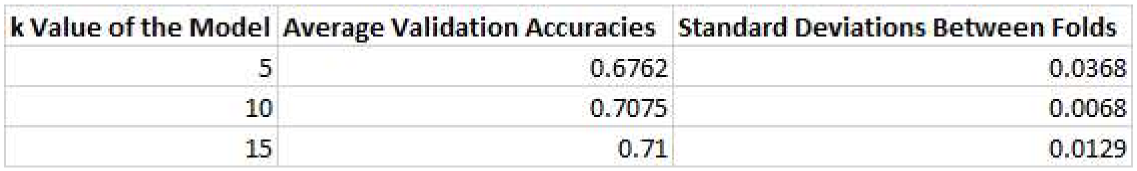
You will get full points from here as long as you have a good (enough) summary of your work, regardless of your best performance or what you have decided to talk about in the last few lines.

Report:

The aim of this homework was to build a k-Nearest Neighbors classifier for German Credit Risk dataset to classify people according to their credit risks by using cross validation on training dataset, testing the best model, also practice feature handling.

In terms of preprocessing, I first analyzed the dataset to understand it. There were NaN values only in "Housing" column of both the train and test set. I imputed these missing values by using the mode of train dataset for both datasets, in order not to gain any advantage for testing. These datasets consists of numerical, ordinal and categorical features. Since k-nn classifier was used for this homework, I transformed all the ordinal and categorical features into numerical ones, by using mapping for ordinal and 'One Hot Encoding' for categorical attributes.

3 different k-nn classifiers were modelled with k values 5, 10 and 15, with all default values for other parameters. 5-fold cross validation was used and I took the average of these cross validation scores for each model. Average validation accuracies and standard deviations between folds for different approaches can be observed in the table below:



k Value of the Model	Average Validation Accuracies	Standard Deviations Between Folds
5	0.6762	0.0368
10	0.7075	0.0068
15	0.71	0.0129

I selected the 15-Nearest Neighbors Classifier, because it gave the best average cross validation score, with 71%. This classifier gave 66.5% classification accuracy on test data.

For the bonus part, when I removed 'Credit Amount' and picked k as 25, the model gave 69% classification accuracy on test data. For another approach, I normalized all the features for scaling and obtained 70.5% accuracy on test set, with k as 15 which is the best model I have found. My other work (which are not displayed in the notebook) such as removing different columns didn't improve the accuracy better than that.

11) Submission

Please submit your **"share link" INLINE in Sucourse submissions**. That is we should be able to click 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.