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 an
# 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-
```

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

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

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

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

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

```
Train data dimensionality:  (800, 13)
Examples from train data:
<bound method NDFrame.sample of     AccountStatus  Duration CreditHistory  ...  Other
0             A14        12           A32  ...               A143    A152    1
1             A11         9           A32  ...               A143    A152    1
2             A11        18           A34  ...               A143    A153    1
3             A11        14           A32  ...               A143    A152    2
4             A14        15           A32  ...               A143    A152    2
..            ...       ...           ...  ...                ...     ...  ...
795           A12         9           A32  ...               A143     NaN    1
796           A14        12           A34  ...               A143     NaN    1
797           A12        36           A32  ...               A143    A153    2
798           A12        12           A32  ...               A143    A152    1
799           A13         9           A32  ...               A143    A152    2

[800 rows x 13 columns]>
```

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

```
<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
Information about train data  None
```

```
train_df.head()
```

	AccountStatus	Duration	CreditHistory	CreditAmount	SavingsAccount	EmploymentS
0	A14	12	A32	2859	A65	
1	A11	9	A32	2136	A61	
2	A11	18	A34	5302	A61	
3	A11	14	A32	8978	A61	
4	A14	15	A32	4623	A62	

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("Number of rows in train set:", len(train_df))
print("Number of rows without nan values in train set:", len(train_df.dropna()))
print("------------------------------------------------------------")
print("Train set columns with NaN:")
print(train_df.isnull().any())
print("------------------------------------------------------------")
print(train_df.isnull().sum())
```

```
Number of rows in train set: 800
Number of rows without nan values in train set: 720
------------------------------------------------------------
Train set columns with NaN:
AccountStatus        False
Duration             False
CreditHistory        False
CreditAmount         False
SavingsAccount       False
EmploymentSince      False
PercentOfIncome      False
PersonalStatus       False
Property             False
Age                  False
OtherInstallPlans    False
Housing               True
dtype: bool
------------------------------------------------------------
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
```

```
print("Number of rows in test set:", len(test_df))
print("Number of rows without nan values in test set:", len(test_df.dropna()))
print("------------------------------------------------------------")
print("Test set columns with NaN:")
print(test_df.isnull().any())
print("------------------------------------------------------------")
print(test_df.isnull().sum())
print("------------------------------------------------------------")
```

```
Number of rows in test set: 200
Number of rows without nan values in test set: 180
------------------------------------------------------------
Test set columns with NaN:
AccountStatus        False
Duration             False
CreditHistory        False
CreditAmount         False
SavingsAccount       False
EmploymentSince      False
PercentOfIncome      False
PersonalStatus       False
Property             False
Age                  False
OtherInstallPlans    False
Housing               True
dtype: bool
------------------------------------------------------------
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
print(train_df['Housing'].value_counts())

mode_train = train_df['Housing'].mode()[0]
print("Mode value of train set:", mode_train)

train_df['Housing'] = train_df['Housing'].fillna(mode_train)
test_df['Housing'] = test_df['Housing'].fillna(mode_train)
```

```
A152    512
A151    130
A153     78
```

```
Name: Housing, dtype: int64
Mode value of train set: A152
```

```
print("Train set columns with NaN after imputing missing data:")
print(train_df.isnull().sum())
print("--------------------------------------")
print("Test set columns with NaN after imputing missing data:")
print(test_df.isnull().sum())
```

```
Train set columns with NaN after imputing missing data:
AccountStatus        0
Duration             0
CreditHistory        0
CreditAmount         0
SavingsAccount       0
EmploymentSince      0
PercentOfIncome      0
PersonalStatus       0
Property             0
Age                  0
OtherInstallPlans    0
Housing              0
dtype: int64
--------------------------------------
Test set columns with NaN after imputing missing data:
AccountStatus        0
Duration             0
CreditHistory        0
CreditAmount         0
SavingsAccount       0
EmploymentSince      0
PercentOfIncome      0
PersonalStatus       0
Property             0
Age                  0
OtherInstallPlans    0
Housing              0
dtype: int64
```

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

map_dict = {'A11':'... < 0 DM', 'A12' : '0 <= ... < 200 DM', 'A13' : '... >= 200 DM / sala
            'A30' : 'no credits taken/all credits paid back duly','A31' : 'all credits at
            'A33' : 'delay in paying off in the past','A34' : 'critical account/other cred
            'A62' : '100 <= ... < 500 DM','A63' : '500 <= ... < 1000 DM','A64' : '.. >= 10
            'A72' : '... < 1 year','A73' : '1 <= ... < 4 years','A74' : '4 <= ... < 7 year
            'A92' : 'female : divorced/separated/married','A93' : 'male : single','A94' :
            'A121' : 'real estate','A122' : 'if not A121 : building society savings agreem
            'A123' : 'if not A121/A122 : car or other, not in attribute 6','A124' : 'unkno
            'A143' : 'none','A151' : 'rent','A152' : 'own','A153' : 'for free'}

train_df = train_df.replace(map_dict)
test_df = test_df.replace(map_dict)



for col in train_df.select_dtypes('object'):
  print(col,train_df[col].unique())
```

```
AccountStatus ['no checking account' '... < 0 DM' '0 <= ... < 200 DM'
 '... >= 200 DM / salary assignments for at least 1 year']
CreditHistory ['existing credits paid back duly till now'
 'critical account/other credits existing (not at this bank)'
 'delay in paying off in the past'
 'all credits at this bank paid back duly'
 'no credits taken/all credits paid back duly']
SavingsAccount ['unknown/ no savings account' '... < 100 DM' '100 <= ... < 500 DM'
 '500 <= ... < 1000 DM' '.. >= 1000 DM']
EmploymentSince ['unemployed' '1 <= ... < 4 years' '.. >= 7 years' '4 <= ... < 7 year
 '... < 1 year']
PersonalStatus ['male : single' 'male : divorced/separated'
 'female : divorced/separated/married' 'male : married/widowed']
Property ['unknown / no property' 'real estate'
 'if not A121 : building society savings agreement/life insurance'
 'if not A121/A122 : car or other, not in attribute 6']
OtherInstallPlans ['none' 'bank' 'stores']
Housing ['own' 'for free' 'rent']
```

```
#Transforming of Ordinal values:
employment_map = {'unemployed':0,'... < 1 year':1,'1 <= ... < 4 years':2,'4 <= ... < 7 yea
train_df['EmploymentSince'] = train_df['EmploymentSince'].replace(employment_map)
test_df['EmploymentSince'] = test_df['EmploymentSince'].replace(employment_map)

account_map = {'no checking account':0, '... < 0 DM':1,'0 <= ... < 200 DM':2,'... >= 200 D
train_df['AccountStatus'] = train_df['AccountStatus'].replace(account_map)
test_df['AccountStatus'] = test_df['AccountStatus'].replace(account_map)
```

```
#Transforming of Categorical values:
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore')
dummies= enc.fit_transform(train_df[['OtherInstallPlans','PersonalStatus','Property','Cred
dummies = pd.DataFrame(dummies)
train_df = pd.merge(train_df,dummies,right_index=True,left_index=True)
train_df = train_df.drop(columns=['OtherInstallPlans','PersonalStatus','Property','CreditH


dummies_test = enc.transform(test_df[['OtherInstallPlans','PersonalStatus','Property','Cre
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=['OtherInstallPlans','PersonalStatus','Property','CreditHis


train_label.head()
```

```
0    1
1    1
2    1
3    2
4    2
Name: Risk, dtype: int64
```

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

# 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)

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

print(accuracies)
```

```
[0.6775, 0.7075, 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
import numpy as np

# Train the best classifier using all training set
best_knn = KNeighborsClassifier(n_neighbors=kVals[np.argmax(accuracies)])

best_knn.fit(train_df, train_label)

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

# Print accuracy of test data
accuracy = accuracy_score(test_label, predictions)
print('Test score for the best model %.2f%% ' % (accuracy*100))
```

```
Test score for the best model 66.50%
```

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.

```
train_df.head()
```

	AccountStatus	Duration	CreditAmount	EmploymentSince	PercentOfIncome	Age	0
0	0	12	2859	0	4	38	0.0
1	1	9	2136	2	3	25	0.0
2	1	18	5302	4	2	36	0.0
3	1	14	8978	4	1	45	0.0
4	0	15	4623	2	3	40	0.0

```
# Remove 'Age' -> it might be irrelevant to target label
train_df_removed =train_df.drop(['Age'], axis=1)
test_df_removed =test_df.drop(['Age'], axis=1)


kVals = [12,13,14,15,16,17]

accuracies = []

for k in kVals:
  # Initialize a k-NN classifier with k neighbors
  knn = KNeighborsClassifier(n_neighbors=k)
  scores = cross_val_score(knn, train_df_removed, train_label, cv=5)

  accuracies.append(mean(scores))


# After removing 'Age' feature, the accuracy increased. --> 66.50% to 67.00%

best_knn = KNeighborsClassifier(n_neighbors=kVals[np.argmax(accuracies)])
best_knn.fit(train_df_removed, train_label)

# Estimate the prediction of the test data
predictions = best_knn.predict(test_df_removed)


# Print accuracy of test data
```

```
accuracy = accuracy_score(test_label, predictions)
print('Test score for the best model %.2f%% ' % (accuracy*100))
```

```
Test score for the best model 67.00%
```

```
# Additional Features -> some columns added and they helped to predict target label, impro
train_df_new = train_df_removed
train_df_new['CreditAmount*PercentOfIncome'] = train_df_removed['CreditAmount']*train_df_r
train_df_new['CreditAmount/Duration'] = train_df_removed['CreditAmount']/train_df_removed[


test_df_new = test_df_removed
test_df_new['CreditAmount*PercentOfIncome'] = test_df_removed['CreditAmount']*test_df_remo
test_df_new['CreditAmount/Duration'] = test_df_removed['CreditAmount']/test_df_removed['Du


# Normalization is done to obtain balance between features

train_df_normalized = (train_df_new - train_df_new.min()) / (train_df_new.max() - train_df
test_df_normalized = (test_df_new - test_df_new.min()) / (test_df_new.max() - test_df_new.
train_df_normalized.head()
```

	AccountStatus	Duration	CreditAmount	EmploymentSince	PercentOfIncome	0	1
0	0.000000	0.117647	0.143557	0.0	1.000000	0.0	1.0
1	0.333333	0.073529	0.103775	0.5	0.666667	0.0	1.0
2	0.333333	0.205882	0.277980	1.0	0.333333	0.0	1.0
3	0.333333	0.147059	0.480247	1.0	0.000000	0.0	1.0
4	0.000000	0.161765	0.240618	0.5	0.666667	0.0	1.0

```
# k-NN classifier after improvements on the dataset

# I tried with more k values
kVals = [12,13,14,15,16,17]

# 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_normalized, train_label, cv=5)

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

```
print(accuracies)
```

```
[0.72, 0.725, 0.7225, 0.73125, 0.72, 0.725]
```

```
best_knn = KNeighborsClassifier(n_neighbors=kVals[np.argmax(accuracies)])
best_knn.fit(train_df_normalized, train_label)

# Estimate the prediction of the test data
predictions = best_knn.predict(test_df_normalized)

# Print accuracy of test data
accuracy = accuracy_score(test_label, predictions)
print('Test score for the best model %.2f%% ' % (accuracy*100))
```

```
Test score for the best model 70.00%
```

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.

The Executive Summary

The problem is trying to train a model and implement German Credit Risk dataset with python. In the end, the aim is to build a k-NN classifier to determine each person as having a good or bad credit, and the highest accuracy score is trying to be achieved with different k values [5,10, 15]. Then, I have predicted the testing data on the best k-NN Classifier so that I can obtain the

possible highest accuracy on the test set. Moreover, a dataset can be further analysed by trying larger k values, scaling input features, removing some features and/or with the help of normalization. So, in the last part of the problem those improvement steps will be tried on the German Credit Risk dataset.

First of all,dataset is loaded and started to be analysed with several features of pandas library to have an idea on the data itself such as shape, info, head etc. Then, the separation of label column is needed to obtain objective results while testing. Due to the possibility of missing NaN values in the dataset, all columns are analysed in terms of NaN values, and 'Housing' column is the only column with missing values after analysis. So, I imputed missing values by replacing with mode value in train and test set. Notice that to impute missing values in test set I have used the mode of train set because otherwise it would cause biased results. In the next step, by using the dictionary for mapping, all meaningless A11,A12.. values are replaced with the meaningful correspondences in terms of string values. However, in k-NN classifier it is necessary to have numberical values to measure the distance between values. So, all features are analysed whether it is ordinal or categorical. Then, depends on their type (ordinal/categorical), I have applied OneHotEncoder or mapping technics to transform into numerical values. Now, we have dataset which is ready to be analysed with k-NN classifier. In the training part, I have obtained three accuracy values with different kVals [5,10,15]: accuracies = [0.6775, 0.7075, 0.71] and selected the best kVal which has highest accuracy on training set (k=15 is obtained as the best with accuracy 0.71). Then I fit the training set with the kVal and predicted using the test set.

Test score for the best model 66.50% (with k=15 as parameter)

However, to obtain better result on the test set, dataset can be analysed further. So, I have applied some technics to improve my test score. First of all, I have removed 'Age' column which seems irrelevant to our target label and it improved my accuracy as I expected. And then I have added some attributes which I found useful to use to predict the target label: CreditAmount*PercentOfIncome and CreditAmount/Duration. Those additional two features improved my predictions on the test set. Moreover, there is obvious imbalance between values of CreditAmount, CreditAmount*PercentOfIncome and CreditAmount/Duration compared to other features because those three columns have larger values as we all expected. So, they dominate the distance measured between data points. So, to avoid this problem I normalized features in a specific range. Finally, I have again trained and tested with different k values but this time I used more kVals on the k-NN classifier. kVals = [12,13,14,15,16,17] and correspondence accuracies obtained: accuracies = [0.72, 0.725, 0.7225, 0.73125, 0.72, 0.725] The best kVal again obtained as k=15 and I fit my training data and tested with test set to obtain accuracy score after all those improvement on dataset. As I expected, improvement on dataset worked and finally,

Test score for the best model 70.00% (with k=15 as parameter)

So, as it can be seen I improved accuracy on test set from 66.50% to 70.00% with parameter k=15. In conclusion, I have obtained much better accuracy in the end with the help of feature removing and addition, feature normalization.

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.