DL-4

March 13, 2021

1 AIQDSC28 – Introduction to deep learning Algorithms Nicola Ronzoni

organization of the work:

The final goal of the project is to develop two neural networks, one with no-text info (numeric variables) and one with no-text info and text info (doctor's reports), that give me the re-hospitalization prediction. I use as target variable the DAYS_NEXT_ADMIT. Since DAYS_NEXT_ADMIT contain a lot of Nan values, I will carefully pre-process them in the first section.

The project have different sections:

1 Load, inspect and filter the data

2 Normalization of the text

3 Treatment of datetime variable

4 Convertion of categorical variables into numeric ones

5 Creation of the dataset for the neural networks

6 Target variable for the neural networks

7 Feature selection for no text neural networks

8 Neural networks for no text

9 Neural network for no text and text info

10 Conclusion and possible future work

1.1 Load, inspect and filter the data

[1]:
```
import pandas as pd
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from matplotlib.colors import ListedColormap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
#import train and test set
df_train = pd.read_csv("http://www.i3s.unice.fr/~riveill/dataset/
 ↪MIMIC-III-readmission/train.csv.zip")
df_test = pd.read_csv("http://www.i3s.unice.fr/~riveill/dataset/
 ↪MIMIC-III-readmission/test.csv.zip")


df_train.head()
```

[2]:
```
   SUBJECT_ID  HADM_ID            ADMITTIME            DISCHTIME  \
0         937   148592  2163-01-20 18:39:00  2163-01-24 08:00:00
1        3016   159142  2107-01-23 02:45:00  2107-01-26 14:00:00
2        2187   186282  2134-06-24 23:30:00  2134-07-02 17:45:00
3       19213   140312  2202-11-02 12:32:00  2202-11-05 14:20:00
4         425   118058  2149-05-13 12:23:00  2149-05-26 20:00:00

   DAYS_NEXT_ADMIT       NEXT_ADMITTIME ADMISSION_TYPE            DEATHTIME  \
0         0.061806  2163-01-24 09:29:00      EMERGENCY  2163-01-26 08:00:00
1              NaN                  NaN      EMERGENCY                  NaN
2              NaN                  NaN      EMERGENCY                  NaN
3        12.968056  2202-11-18 13:34:00      EMERGENCY                  NaN
4              NaN                  NaN      EMERGENCY                  NaN

         DISCHARGE_LOCATION INSURANCE  … mental misc muscular neoplasms  \
0             DEAD/EXPIRED  Medicare  …      0    0        0         0
1         HOME HEALTH CARE  Medicare  …      2    0        0         0
2  REHAB/DISTINCT PART HOSP  Medicaid  …      1    2        1         0
3                      HOME  Medicare  …      0    0        0         0
4         HOME HEALTH CARE  Medicare  …      0    0        0         0

  nervous pregnancy  prenatal  respiratory  skin  OUTPUT_LABEL
0       1         0         0            0     0             1
1       0         0         0            1     0             0
2       3         0         0            4     0             0
3       0         0         0            1     1             1
4       0         0         0            2     1             0

[5 rows x 34 columns]
```

[3]:
```
#shape of the train and test set
df_train.shape, df_test.shape
```

[3]: ((2000, 34), (901, 34))

[4]:
```
#Number of non-Nan values per variable in train test
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 34 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   SUBJECT_ID          2000 non-null   int64
 1   HADM_ID             2000 non-null   int64
 2   ADMITTIME           2000 non-null   object
 3   DISCHTIME           2000 non-null   object
 4   DAYS_NEXT_ADMIT     1210 non-null   float64
 5   NEXT_ADMITTIME      1210 non-null   object
 6   ADMISSION_TYPE      2000 non-null   object
 7   DEATHTIME           158 non-null    object
 8   DISCHARGE_LOCATION  2000 non-null   object
 9   INSURANCE           2000 non-null   object
 10  MARITAL_STATUS      1924 non-null   object
 11  ETHNICITY           2000 non-null   object
 12  DIAGNOSIS           1998 non-null   object
 13  TEXT                1925 non-null   object
 14  GENDER              2000 non-null   object
 15  DOB                 2000 non-null   object
 16  blood               2000 non-null   int64
 17  circulatory         2000 non-null   int64
 18  congenital          2000 non-null   int64
 19  digestive           2000 non-null   int64
 20  endocrine           2000 non-null   int64
 21  genitourinary       2000 non-null   int64
 22  infectious          2000 non-null   int64
 23  injury              2000 non-null   int64
 24  mental              2000 non-null   int64
 25  misc                2000 non-null   int64
 26  muscular            2000 non-null   int64
 27  neoplasms           2000 non-null   int64
 28  nervous             2000 non-null   int64
 29  pregnancy           2000 non-null   int64
 30  prenatal            2000 non-null   int64
 31  respiratory         2000 non-null   int64
 32  skin                2000 non-null   int64
 33  OUTPUT_LABEL        2000 non-null   int64
dtypes: float64(1), int64(20), object(13)
memory usage: 531.4+ KB
```

Since I would like to predict re-hospitalization, I delete observation of death patient in order to avoid data leakage. I would like to obtain neural networks that works with alive patients. If I train the neural network considering also death patients this additional information can allow the model to learn or know something that it otherwise would not know.

[5]:
```
# keep patients in which Deathtime equal to Nan in train and test set
df_train=df_train[df_train['DEATHTIME'].isna()]
df_test=df_test[df_test['DEATHTIME'].isna()]
```

[6]:
```
df_train.shape, df_test.shape
```

[6]:
```
((1842, 34), (843, 34))
```

I deleted 158 observation from the train set and 58 observation from the test set

Another thing to do in order to develop a good model is to delete the observation of patient that are re-hospitalized after more than one year. If I look at the distribution of the DAYS_NEXT_ADMIT I have more than 90% of the re-hospitalization before one year of time. After one year it is possible that the past medical history of the patient affect less the re-hospitalization. In addition exogenous variable can influence the final output (car accident, broken leg ...)

[7]:
```
#look at the percentile of DAYS_NEXT_ADMIT in train and test set
df_train['DAYS_NEXT_ADMIT'].quantile(0.93), df_test['DAYS_NEXT_ADMIT'].
 ↪quantile(0.94)

#more than 90% of re-hospitalization are before one year time
```

[7]:
```
(422.82447222222225, 363.7695833333322)
```

[8]:
```
#delete observation of the patient for which the time passed between the last␣
 ↪two hospitalization is greater than one year
#train
df_train= df_train[~(df_train['DAYS_NEXT_ADMIT'] >365)]
#test
df_test= df_test[~(df_test['DAYS_NEXT_ADMIT'] > 365)]
```

[9]:
```
df_train.shape, df_test.shape
```

[9]:
```
((1748, 34), (811, 34))
```

I deleted 94 observations from the train set and 32 observations from the test

In the variable DAYS_NEXT_ADMIT the Nan values are related to a specific value which is supposed to be obtained (number of day before the next admission greater than patients which are re-hospitalized OUTPUT_LABEL=1). I deal with Missing not at random data, that are the most problematic ones. First look at the ratio of Nan values in the train and test set

[10]:
```
#ratio of Nan in DAYS_NEXT_ADMIT train set
ratio_train = np.sum(df_train['DAYS_NEXT_ADMIT'].isna())/
 ↪len(df_train['DAYS_NEXT_ADMIT'])
```

[11]:
```
ratio_train
```

[11]:
```
0.3707093821510298
```

[12]:
```
#ratio of Nan in DAYS_NEXT_ADMIT test set
ratio_test = np.sum(df_test['DAYS_NEXT_ADMIT'].isna())/
 ↪len(df_test['DAYS_NEXT_ADMIT'])
```

[13]:
```
ratio_test
```

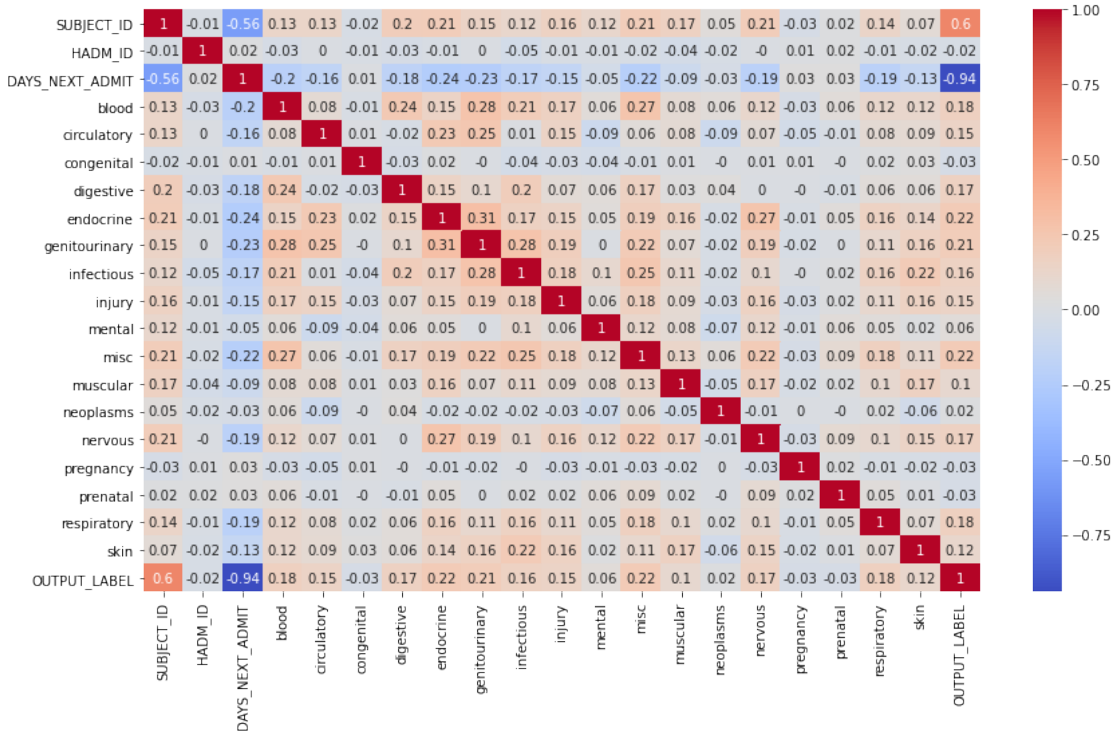
[13]: 0.3970406905055487

In both cases, train and test set, a large portion of the data is missing. In addition I deal with Missing not at random, I exclude to delete patients with DAYS_NEXT_ADMIT=Nan. They represent a particular subset of the target. If I delete this observations I reduce variability on the final output and this could afflict negatively the Neural Networks.
I know that missing values on the variables DAYS_NEXT_ADMIT are positive related with a high number of days before the next re-hospitalization. I supppose that patients that are not re-hospitalized have a number of days next to admit higher than patients that are been re-hospitalized (OUTPUT_LABEL=1).I can substitute missing values with 365 days, that is my cut off point. By doing that I restrict the area of the prediction to one year after the last hospitalization, but I am going to predict also days next to admit for non re-hospitalized patients (OUTPUT_LABEL=0)

[14]:
```
#replace Nan value with the max 365
df_train['DAYS_NEXT_ADMIT'] = df_train['DAYS_NEXT_ADMIT'].replace(np.nan, 365)
df_test['DAYS_NEXT_ADMIT'] = df_test['DAYS_NEXT_ADMIT'].replace(np.nan, 365)
```

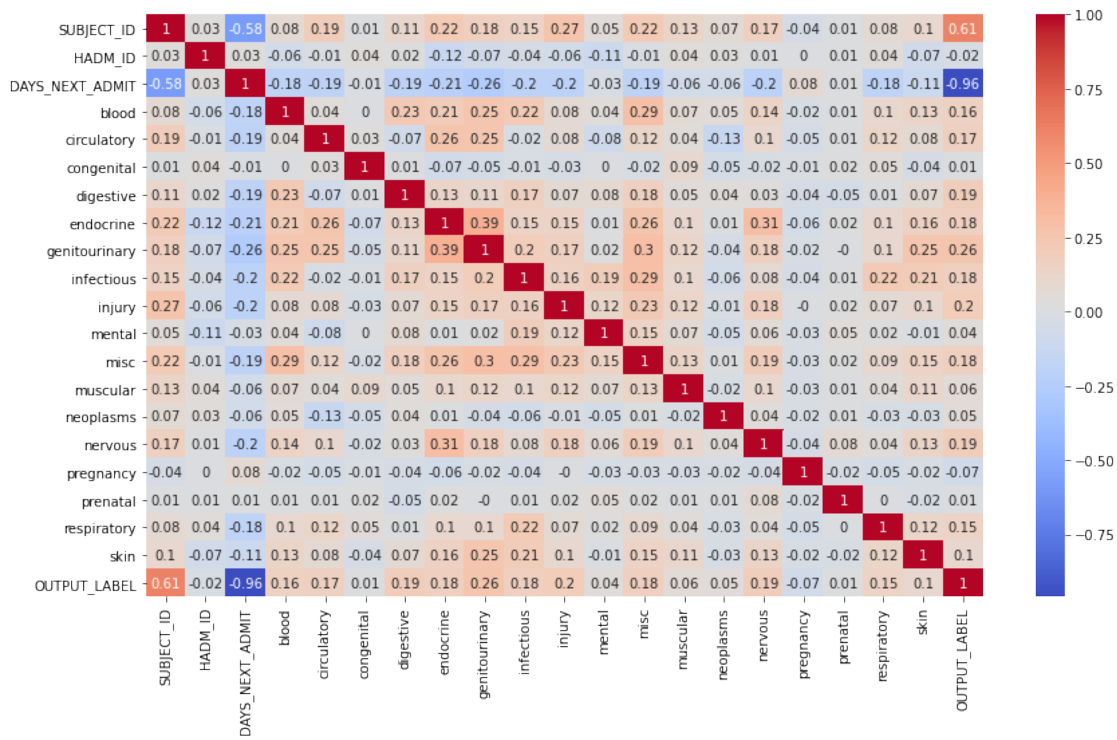
[15]:
```
#plot the correlation of independent variables
# Plot the correlation heatmap
from termcolor import colored as cl
plt.figure(figsize=(14, 8))
corr_matrix = df_train.corr().round(2)
sns.heatmap(data=corr_matrix,cmap='coolwarm',annot=True)
```

[15]: <AxesSubplot:>



```
[16]: plt.figure(figsize=(14, 8))
      corr_matrix = df_test.corr().round(2)
      sns.heatmap(data=corr_matrix,cmap='coolwarm',annot=True)
```

```
[16]: <AxesSubplot:>
```



If I look at the correlation map, I can see a high negative correlation between OUTPUT_LABEL and DAYS_NEXT_ADMIT. This make sense: patients with OUTPUT_LABEL =1 have a DAYS_NEXT_ADMIT small, instead patients with OUTPUT_LABEL=0 have a DAYS_NEXT_ADMIT big.

df_train and df_test will be sets in which i will put all new variable that I create. After that i will select careful the most important ones

1.2 Normalization of the text

By normalizing the reports, I attempt to reduce the randomness in it, bringing it closer to a predefined "standard". This helps into reducing the amount of different information that the computer has to deal, and therefore improves efficiency. In addition I would like preserve information with high variability in order to explain my target variable.

At the end of this stage i will upload the new texts treated in df_train and df_test

In order to compute the normalization I select only the column of text from the train and test sets

[17]:
```
#test

text_test=df_test['TEXT']

text_test.dtypes, text_test.shape
```

[17]:
```
(dtype('O'), (811,))
```

[18]:
```
#train

text_train=df_train['TEXT']
text_train.dtypes, text_train.shape
```

[18]:
```
(dtype('O'), (1748,))
```

Convertion of Upper case in lower case in the text_train and text_test arrays

[19]:
```
#convert text in lower case
import nltk.corpus
from nltk.corpus import stopwords
#nltk.download('stopwords')
from nltk.tokenize import word_tokenize
#test
text_test_lw =[]
for text in text_test:
    text_tokens_train=word_tokenize(str(text))
    lower_train=' '.join([w.lower() for w in text_tokens_train])
    text_test_lw.append(lower_train)
#train
text_train_lw =[]
for text in text_train:
    text_tokens_train=word_tokenize(str(text))
    lower_train=' '.join([w.lower() for w in text_tokens_train])
    text_train_lw.append(lower_train)
```

Remove punctuation sign: This step is really important, in doctor's report there is a predefined layout with a lot of sign such as: [ ] * , . ; :

[20]:
```
#remove some punctuation sign

import string
#test
test_pc =[]
for text in text_test_lw:
    text_tokens_train=word_tokenize(str(text))
    punctual_train=' '.join([w.translate(str.maketrans('', '', string.
↪punctuation)) for w in text_tokens_train])
    test_pc.append(punctual_train)
#train
train_pc =[]
for text in text_train_lw:
    text_tokens_train=word_tokenize(str(text))
    punctual_train=' '.join([w.translate(str.maketrans('', '', string.
↪punctuation)) for w in text_tokens_train])
```

```
    train_pc.append(punctual_train)
```

Stemming: Crude heuristic process that cuts off the end of words in the hope of achieving a reduction in the forms of a word.

[21]:
```
#stemming
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
stemmer = PorterStemmer()
#test
test_stem=[]
for text in test_pc:
    text_tokens_train=word_tokenize(str(text))
    stemmatized_train=' '.join([stemmer.stem(w) for w in text_tokens_train])
    test_stem.append(stemmatized_train)
#train
train_stem=[]
for text in train_pc:
    text_tokens_train=word_tokenize(str(text))
    stemmatized_train=' '.join([stemmer.stem(w) for w in text_tokens_train])
    train_stem.append(stemmatized_train)
```

Remove number: Date of birth, admission, dismission are present in clinical reports. They involve a lot of numbers, and in addition I have already this information in numerical variables.

[22]:
```
#remove number

import re
#test
test_nu =[]
for text in test_stem:
    text_tokens_train=word_tokenize(str(text))
    nonum_train=' '.join([re.sub(r'\d+', '', w) for w in text_tokens_train])
    test_nu.append(nonum_train)
#train
train_nu =[]
for text in train_stem:
    text_tokens_train=word_tokenize(str(text))
    nonum_train=' '.join([re.sub(r'\d+', '', w) for w in text_tokens_train])
    train_nu.append(nonum_train)
```

Remove words that appears less than 5 times.The Majority of report's have more than 1000 words. With this passage I delete useless information

[23]:
```
#remove words that appears less than 5 times
from collections import Counter

#train
train_k=[]
for text in train_nu:
    text_tokens_train=word_tokenize(str(text))
    counted=Counter(text_tokens_train)
    k_train=[el for el in text_tokens_train if text_tokens_train.count(el) >= 5]
    more_train=' '.join(k_train)
    train_k.append(more_train)
#test
test_k=[]
for text in test_nu:
    text_tokens_test=word_tokenize(str(text))
    counted=Counter(text_tokens_test)
    k_test=[el for el in text_tokens_test if text_tokens_test.count(el) >= 5]
    more_test=' '.join(k_test)
    test_k.append(more_test)
```

Remove stop words, of course I can add other words at the default ones.

Stop word by default delete: 'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any', 'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not', 'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where', 'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom', 't', 'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it', 'how', 'further', 'was', 'here', 'than'.

Of course every report has a pre-filled intestation in which words that are not included in the list above are repeated a lot of time such as: 'patient','tablet','name','discharg','sig','histori','admiss','date','namepattern','note','am','pm','telephonefax','m','f','m

In addition I delete all numbers so unit measurement of medicine are useless: 'per','day','mg','md','daili','x','ml'

[24]:
```
#remove english stop word
#creation of the set of stopwords
stopwords = nltk.corpus.stopwords.words('english')
newStopWords =␣
 ↪['patient','tablet','name','discharg','sig','histori','admiss','date','namepattern','note',
stopwords.extend(newStopWords)
#test
test_sw = []
for text in test_k:
```

```
    text_tokens_train = word_tokenize(str(text))
    tokens_without_sw_train = [word for word in text_tokens_train if not word␣
 ↪in stopwords]
    tokens_without_sw_s_train = ' '.join(tokens_without_sw_train)
    test_sw.append(tokens_without_sw_s_train)
#train
train_sw = []
for text in train_k:
    text_tokens_train = word_tokenize(str(text))
    tokens_without_sw_train = [word for word in text_tokens_train if not word␣
 ↪in stopwords]
    tokens_without_sw_s_train = ' '.join(tokens_without_sw_train)
    train_sw.append(tokens_without_sw_s_train)
```

Append the new variable to the dataset

[25]:
```
#append the new var to the dataframe
#train
df_train.insert(loc=1,column='text',value=train_sw)
```

[26]:
```
#append the new var to the dataframe
#test
df_test.insert(loc=1,column='text',value=test_sw)
```

1.3 Treatment of datetime variable

In this section I will treat variables in datetime format, in particular I will deal with: admit time, discharge time and date of birth of the patient.

I suppose that the time passed in hospital is positive correlated with the probability of being re-hospitalize. More time a patient passed in hospital more probability this patient has to be re-hospitalized. So I create a new numeric variable for the time passed in hospital during the last hospitalization.

[27]:
```
#create a new variable recovery for the amount of time passed in hospital.



import datetime
#train
admittime_train = pd.to_datetime(df_train['ADMITTIME'])
discharge_train =pd.to_datetime(df_train['DISCHTIME'])
recovery_time_train=discharge_train-admittime_train
#test
admittime_test= pd.to_datetime(df_test['ADMITTIME'])
discharge_test=pd.to_datetime(df_test['DISCHTIME'])
recovery_time_test=discharge_test-admittime_test
```

For the time passed in hospital I use only days. I am going to convert the format.

[28]:
```
#change the format of the date into days
recovery_time_train=recovery_time_train.dt.days
recovery_time_test=recovery_time_test.dt.days
```

[29]:
```
recovery_time_test
```

[29]:
```
0      39
1       6
2       4
3       6
4       9
       ..
896     3
897    11
898    15
899     6
900     2
Length: 811, dtype: int64
```

[30]:
```
recovery_time_train
```

[30]:
```
1        3
2        7
3        3
4       13
5        6
        ..
1995     4
1996     8
1997     8
1998     5
1999    11
Length: 1748, dtype: int64
```

For the variable date of birth I am going to consider only the year of birth, since this add enough variability.

[31]:
```
#treatment of date of birth only take the year
#train
dateofbirth_train=pd.to_datetime(df_train['DOB'])
#test
dateofbirth_test=pd.to_datetime(df_test['DOB'])
```

[32]:
```
dateofbirth_train=dateofbirth_train.dt.year
dateofbirth_test=dateofbirth_test.dt.year
```

1.4 Convertion of categorical variables in numeric ones

In This section I convert categorical variables into numeric ones.

Label encoding is simply converting each value in a column to a number. This work for the gender variable, because it is a nominal variable.

[33]:
```
#train
df_train["GENDER"] = df_train["GENDER"].astype('category')
df_train["gender_cat"] = df_train["GENDER"].cat.codes
#test
df_test["GENDER"] = df_test["GENDER"].astype('category')
df_test["gender_cat"] = df_test["GENDER"].cat.codes
```

Label encoding has the advantage that it is straightforward but it has the disadvantage that the numeric values can be "misinterpreted" by the algorithms. The distance between two possible value can not correspond to the real distance in life. For Example in the variable DISCHARGE_LOCATION home could be set =1, long term care hospital =2 and short term hospital =3 but indeed I know that short term hospital is more close to home than the long term. The alternative adopted, to treat ordinal variables, is to create a number of dummy variables (0/1) equal to the number of possible categories - 1 to avoid multicollinearity.

[34]:
```
#generate dummy variables for each categorical variable. I am going to use a␣
↪prefix in order to identify them better in the next step.
#train
df_train=pd.get_dummies(df_train, columns=["ADMISSION_TYPE",␣
↪"INSURANCE","MARITAL_STATUS","ETHNICITY","DISCHARGE_LOCATION",],␣
↪prefix=["ADM", "INS","MAR","ETH","DIS"])
#test
df_test=pd.get_dummies(df_test, columns=["ADMISSION_TYPE",␣
↪"INSURANCE","MARITAL_STATUS","ETHNICITY","DISCHARGE_LOCATION",],␣
↪prefix=["ADM", "INS","MAR","ETH","DIS"])
```

Drop one categories from each original categorical variable to avoid multicollinearity. Doesn't matter which one, in any case the remaining dummies preserve the information.

[35]:
```
#train
df_train=df_train.
↪drop(columns=["ADM_URGENT","INS_Private","MAR_SEPARATED","ETH_ASIAN","DIS_HOME"])
#test
df_test=df_test.
↪drop(columns=["ADM_URGENT","INS_Private","MAR_SEPARATED","ETH_ASIAN","DIS_HOME"])
```

[36]:
```
#print all variables that I have right now
df_train.columns
```

[36]:
```
Index(['SUBJECT_ID', 'text', 'HADM_ID', 'ADMITTIME', 'DISCHTIME',
       'DAYS_NEXT_ADMIT', 'NEXT_ADMITTIME', 'DEATHTIME', 'DIAGNOSIS', 'TEXT',
       'GENDER', 'DOB', 'blood', 'circulatory', 'congenital', 'digestive',
```

```
 'endocrine', 'genitourinary', 'infectious', 'injury', 'mental', 'misc',
 'muscular', 'neoplasms', 'nervous', 'pregnancy', 'prenatal',
 'respiratory', 'skin', 'OUTPUT_LABEL', 'gender_cat', 'ADM_ELECTIVE',
 'ADM_EMERGENCY', 'INS_Government', 'INS_Medicaid', 'INS_Medicare',
 'INS_Self Pay', 'MAR_DIVORCED', 'MAR_MARRIED', 'MAR_SINGLE',
 'MAR_UNKNOWN (DEFAULT)', 'MAR_WIDOWED', 'ETH_BLACK/AFRICAN AMERICAN',
 'ETH_HISPANIC/LATINO', 'ETH_OTHER/UNKNOWN', 'ETH_WHITE',
 'DIS_DISC-TRAN CANCER/CHLDRN H', 'DIS_DISCH-TRAN TO PSYCH HOSP',
 'DIS_HOME HEALTH CARE', 'DIS_HOME WITH HOME IV PROVIDR',
 'DIS_HOSPICE-HOME', 'DIS_HOSPICE-MEDICAL FACILITY', 'DIS_ICF',
 'DIS_LEFT AGAINST MEDICAL ADVI', 'DIS_LONG TERM CARE HOSPITAL',
 'DIS_OTHER FACILITY', 'DIS_REHAB/DISTINCT PART HOSP',
 'DIS_SHORT TERM HOSPITAL', 'DIS_SNF'],
dtype='object')
```

[37]:
```
#check if the number of variable in the train and test set
df_train.shape, df_test.shape
```

[37]:
```
((1748, 59), (811, 59))
```

1.5 Creation of the dataset for the neural networks

In this section I sum up all variable previously created and compute the last part of pre-treatment. I will select all new dummy variables instead of the categorical ones, the normalized text variable and of course the numeric variables already present in the initial dataset.

[38]:
```
#define a numeric dataset
#not select deathtime column because it is an entire column of null

#train
df_train1=df_train[['gender_cat', 'ADM_EMERGENCY','ADM_ELECTIVE',␣
↪'ETH_WHITE','ETH_OTHER/UNKNOWN','ETH_BLACK/AFRICAN AMERICAN','ETH_HISPANIC/
↪LATINO', 'INS_Medicare','INS_Medicaid','INS_Government','INS_Self Pay',␣
↪'MAR_MARRIED','MAR_SINGLE','MAR_WIDOWED','MAR_DIVORCED','DIS_HOME HEALTH␣
↪CARE','DIS_SNF','DIS_REHAB/DISTINCT PART HOSP','DIS_LONG TERM CARE␣
↪HOSPITAL','DIS_DISC-TRAN CANCER/CHLDRN H','DIS_LEFT AGAINST MEDICAL␣
↪ADVI','DIS_SHORT TERM HOSPITAL','DIS_HOME WITH HOME IV␣
↪PROVIDR','DIS_DISCH-TRAN TO PSYCH HOSP','DIS_ICF','DIS_HOSPICE-MEDICAL␣
↪FACILITY','DIS_HOSPICE-HOME', 'blood', 'circulatory', 'congenital',␣
↪'digestive', 'endocrine', 'genitourinary', 'infectious', 'injury', 'mental',␣
↪'misc', 'muscular', 'neoplasms', 'nervous', 'pregnancy', 'prenatal',␣
↪'respiratory', 'skin','DAYS_NEXT_ADMIT','text']]
df_train1.insert(loc=1,column='recovery_day',value=recovery_time_train)
df_train1.insert(loc=2,column='dob',value=dateofbirth_train)
#test
```

```
df_test1=df_test[['gender_cat', 'ADM_EMERGENCY','ADM_ELECTIVE',␣
↪'ETH_WHITE','ETH_OTHER/UNKNOWN','ETH_BLACK/AFRICAN AMERICAN','ETH_HISPANIC/
↪LATINO', 'INS_Medicare','INS_Medicaid','INS_Government','INS_Self Pay',␣
↪'MAR_MARRIED','MAR_SINGLE','MAR_WIDOWED','MAR_DIVORCED','DIS_HOME HEALTH␣
↪CARE','DIS_SNF','DIS_REHAB/DISTINCT PART HOSP','DIS_LONG TERM CARE␣
↪HOSPITAL','DIS_DISC-TRAN CANCER/CHLDRN H','DIS_LEFT AGAINST MEDICAL␣
↪ADVI','DIS_SHORT TERM HOSPITAL','DIS_HOME WITH HOME IV␣
↪PROVIDR','DIS_DISCH-TRAN TO PSYCH HOSP','DIS_ICF','DIS_HOSPICE-MEDICAL␣
↪FACILITY','DIS_HOSPICE-HOME', 'blood', 'circulatory', 'congenital',␣
↪'digestive', 'endocrine', 'genitourinary', 'infectious', 'injury', 'mental',␣
↪'misc', 'muscular', 'neoplasms', 'nervous', 'pregnancy', 'prenatal',␣
↪'respiratory', 'skin','DAYS_NEXT_ADMIT','text']]
df_test1.insert(loc=1,column='recovery_day',value=recovery_time_train)
df_test1.insert(loc=2,column='dob',value=dateofbirth_test)
```

in This last step I drop all rows that contain null.

[39]:
```
#delete obs for which there is at least a null value in a column.

#train
df_train1 = df_train1.dropna(how='any',axis=0)
#test
df_test1= df_test1.dropna(how='any',axis=0)
```

[40]:
```
df_train1.shape,df_test1.shape
```

[40]:
```
((1748, 48), (721, 48))
```

[41]:
```
df_train1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1748 entries, 1 to 1999
Data columns (total 48 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   gender_cat                  1748 non-null   int8
 1   recovery_day                1748 non-null   int64
 2   dob                         1748 non-null   int64
 3   ADM_EMERGENCY               1748 non-null   uint8
 4   ADM_ELECTIVE                1748 non-null   uint8
 5   ETH_WHITE                   1748 non-null   uint8
 6   ETH_OTHER/UNKNOWN           1748 non-null   uint8
 7   ETH_BLACK/AFRICAN AMERICAN  1748 non-null   uint8
 8   ETH_HISPANIC/LATINO         1748 non-null   uint8
 9   INS_Medicare                1748 non-null   uint8
 10  INS_Medicaid                1748 non-null   uint8
 11  INS_Government              1748 non-null   uint8
 12  INS_Self Pay                1748 non-null   uint8
```

```
 13  MAR_MARRIED                    1748 non-null   uint8
 14  MAR_SINGLE                     1748 non-null   uint8
 15  MAR_WIDOWED                    1748 non-null   uint8
 16  MAR_DIVORCED                   1748 non-null   uint8
 17  DIS_HOME HEALTH CARE           1748 non-null   uint8
 18  DIS_SNF                        1748 non-null   uint8
 19  DIS_REHAB/DISTINCT PART HOSP   1748 non-null   uint8
 20  DIS_LONG TERM CARE HOSPITAL    1748 non-null   uint8
 21  DIS_DISC-TRAN CANCER/CHLDRN H  1748 non-null   uint8
 22  DIS_LEFT AGAINST MEDICAL ADVI  1748 non-null   uint8
 23  DIS_SHORT TERM HOSPITAL        1748 non-null   uint8
 24  DIS_HOME WITH HOME IV PROVIDR  1748 non-null   uint8
 25  DIS_DISCH-TRAN TO PSYCH HOSP   1748 non-null   uint8
 26  DIS_ICF                        1748 non-null   uint8
 27  DIS_HOSPICE-MEDICAL FACILITY   1748 non-null   uint8
 28  DIS_HOSPICE-HOME               1748 non-null   uint8
 29  blood                          1748 non-null   int64
 30  circulatory                    1748 non-null   int64
 31  congenital                     1748 non-null   int64
 32  digestive                      1748 non-null   int64
 33  endocrine                      1748 non-null   int64
 34  genitourinary                  1748 non-null   int64
 35  infectious                     1748 non-null   int64
 36  injury                         1748 non-null   int64
 37  mental                         1748 non-null   int64
 38  misc                           1748 non-null   int64
 39  muscular                       1748 non-null   int64
 40  neoplasms                      1748 non-null   int64
 41  nervous                        1748 non-null   int64
 42  pregnancy                      1748 non-null   int64
 43  prenatal                       1748 non-null   int64
 44  respiratory                    1748 non-null   int64
 45  skin                           1748 non-null   int64
 46  DAYS_NEXT_ADMIT                1748 non-null   float64
 47  text                           1748 non-null   object
dtypes: float64(1), int64(19), int8(1), object(1), uint8(26)
memory usage: 346.5+ KB
```

1.6 Definition of the target for the neural networks

DAYS_NEXT_ADMIT is my target. In the first section i treat Nan values.

[42]:
```
#define the target
#RE-HOSPITALIZATION = YES/NO

y_train=df_train1['DAYS_NEXT_ADMIT']
y_test=df_test1['DAYS_NEXT_ADMIT']
```

[43]:
```
y_test.shape, y_train.shape
```

[43]:
```
((721,), (1748,))
```

I scale the output, in order to fit the neural networks. The activation function that i will used for the output of the neural network is sigmoid so the output variables must be between [0,1]. Once i wil train the neural network and compute the predictions on the test set i can decide to evaluate the MSE on the scaled output or come back and asses the metric on the number of days by using scaler.inverse_transform(scaled_data)

[44]:
```
from sklearn.preprocessing import MinMaxScaler,StandardScaler
```

[45]:
```
scaler=MinMaxScaler()
```

[46]:
```
#output train/test scaled
y_train=pd.DataFrame(y_train)
y_test=pd.DataFrame(y_test)
y_train_scaled=scaler.fit_transform(y_train)
y_test_scaled=scaler.fit_transform(y_test)
```

1.7 Feature selection for no text neural networks

Before implementing the feature selection, I will put together all numeric features, excluding the text .

[47]:

```
X_train=df_train1[['gender_cat', 'ADM_EMERGENCY','ADM_ELECTIVE',␣
↪'ETH_WHITE','ETH_OTHER/UNKNOWN','ETH_BLACK/AFRICAN AMERICAN','ETH_HISPANIC/
↪LATINO', 'INS_Medicare','INS_Medicaid','INS_Government','INS_Self Pay',␣
↪'MAR_MARRIED','MAR_SINGLE','MAR_WIDOWED','MAR_DIVORCED','DIS_HOME HEALTH␣
↪CARE','DIS_SNF','DIS_REHAB/DISTINCT PART HOSP','DIS_LONG TERM CARE␣
↪HOSPITAL','DIS_DISC-TRAN CANCER/CHLDRN H','DIS_LEFT AGAINST MEDICAL␣
↪ADVI','DIS_SHORT TERM HOSPITAL','DIS_HOME WITH HOME IV␣
↪PROVIDR','DIS_DISCH-TRAN TO PSYCH HOSP','DIS_ICF','DIS_HOSPICE-MEDICAL␣
↪FACILITY','DIS_HOSPICE-HOME', 'blood', 'circulatory', 'congenital',␣
↪'digestive', 'endocrine', 'genitourinary', 'infectious', 'injury', 'mental',␣
↪'misc','muscular', 'neoplasms', 'nervous', 'pregnancy', 'prenatal',␣
↪'respiratory', 'skin','recovery_day','dob']]
X_test=df_test1[[ 'gender_cat', 'ADM_EMERGENCY','ADM_ELECTIVE',␣
↪'ETH_WHITE','ETH_OTHER/UNKNOWN','ETH_BLACK/AFRICAN AMERICAN','ETH_HISPANIC/
↪LATINO', 'INS_Medicare','INS_Medicaid','INS_Government','INS_Self Pay',␣
↪'MAR_MARRIED','MAR_SINGLE','MAR_WIDOWED','MAR_DIVORCED','DIS_HOME HEALTH␣
↪CARE','DIS_SNF','DIS_REHAB/DISTINCT PART HOSP','DIS_LONG TERM CARE␣
↪HOSPITAL','DIS_DISC-TRAN CANCER/CHLDRN H','DIS_LEFT AGAINST MEDICAL␣
↪ADVI','DIS_SHORT TERM HOSPITAL','DIS_HOME WITH HOME IV␣
↪PROVIDR','DIS_DISCH-TRAN TO PSYCH HOSP','DIS_ICF','DIS_HOSPICE-MEDICAL␣
↪FACILITY','DIS_HOSPICE-HOME','blood', 'circulatory', 'congenital',␣
↪'digestive', 'endocrine', 'genitourinary', 'infectious', 'injury', 'mental',␣
↪'misc','muscular', 'neoplasms', 'nervous', 'pregnancy', 'prenatal',␣
↪'respiratory', 'skin','recovery_day','dob']]
```

[48]:
```
X_test.shape, X_train.shape
```

[48]: ((721, 46), (1748, 46))

[49]:
```
np.seterr(divide='ignore', invalid='ignore')
```

[49]: {'divide': 'warn', 'over': 'warn', 'under': 'ignore', 'invalid': 'warn'}

Supervised feature selection techniques use the target variable, such as methods that remove irrelevant variables. Feature selection methods use statistical techniques to evaluate the relationship between each input variable and the target variable, and these scores are used as the basis to choose (filter) those input variables that will be used in the model. I have a regression problem since the output is a numeric variable therofore I could evaluate the importance of the features using Pearson's Correlation Coefficient. I can do that because I have test and train separated from the very beginning, otherwise in case of k-fold validation this could be dangerous.

[50]:
```
# load and summarize the dataset
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest , f_regression, f_classif
# configure to select all features
```

```
fs = SelectKBest(score_func=f_regression, k='all')
# learn relationship from training data
fs.fit(X_train, y_train)
# transform train input data
X_train_fs = fs.transform(X_train)
# transform test input data
X_test_fs = fs.transform(X_test)
X_selected = fs.fit_transform(X_train, y_train)
print(X_selected.shape)
```

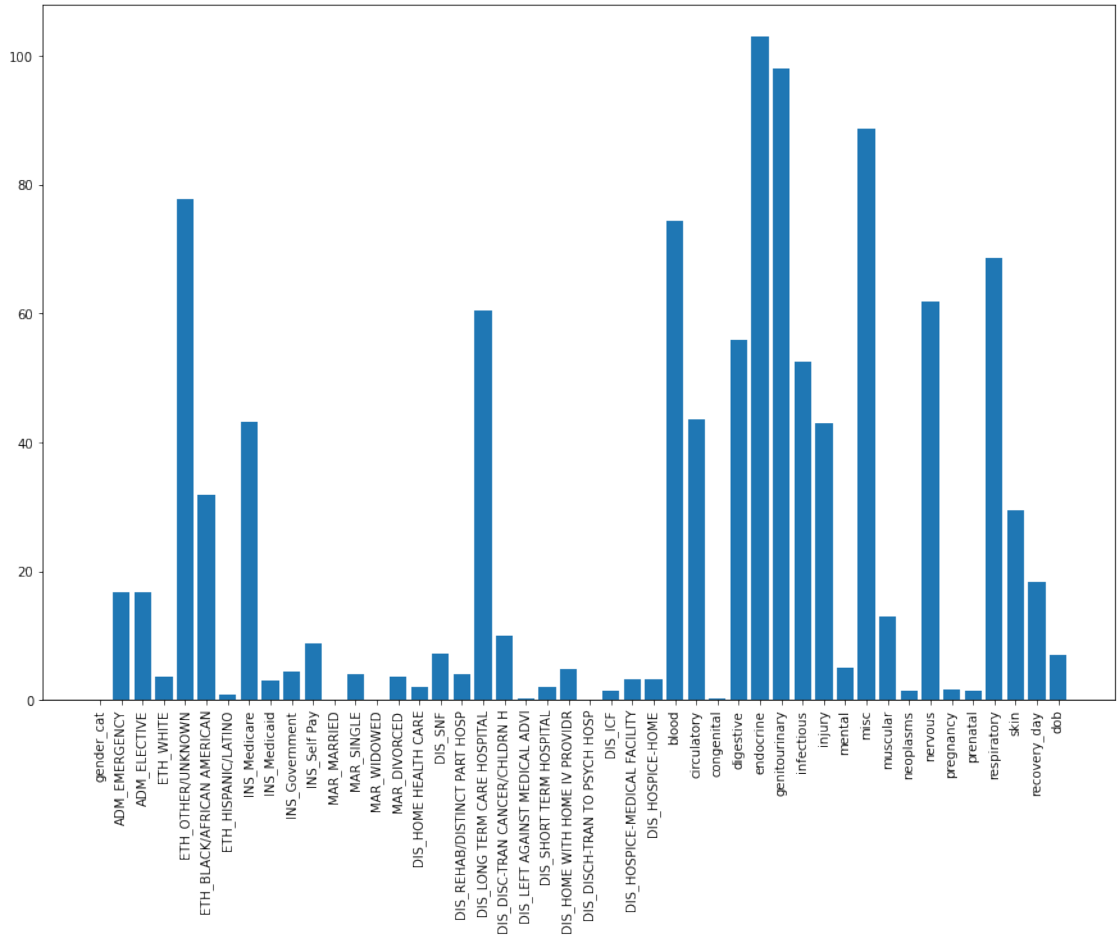
```
(1748, 46)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:72:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(**kwargs)
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:72:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(**kwargs)
```

[51]:
```
# what are scores for the features
for i in range(len(X_test.columns)):
    print( X_test.columns[i], fs.scores_[i])
# plot the scores
plt.figure(figsize=(15,10))
plt.bar([i for i in range(len(X_test.columns))], fs.scores_)
plt.xticks(range(len(X_test.columns)),X_test.columns, rotation=90)
plt.show()
```

```
gender_cat 0.018337122930140493
ADM_EMERGENCY 16.7416571895023
ADM_ELECTIVE 16.7760283395656
ETH_WHITE 3.698223841552072
ETH_OTHER/UNKNOWN 77.70636956475492
ETH_BLACK/AFRICAN AMERICAN 31.790871420889147
ETH_HISPANIC/LATINO 0.7531241774480377
INS_Medicare 43.15896623325177
INS_Medicaid 3.0438333543347063
INS_Government 4.451302772202489
INS_Self Pay 8.811789278019152
MAR_MARRIED 0.08847454478324601
MAR_SINGLE 4.100861672909067
MAR_WIDOWED 0.00457178108299801
MAR_DIVORCED 3.587818394659889
DIS_HOME HEALTH CARE 2.118703950610359
```

```
DIS_SNF 7.243940336256734
DIS_REHAB/DISTINCT PART HOSP 4.015558025871024
DIS_LONG TERM CARE HOSPITAL 60.444444696610674
DIS_DISC-TRAN CANCER/CHLDRN H 10.018420517283316
DIS_LEFT AGAINST MEDICAL ADVI 0.31378529018454476
DIS_SHORT TERM HOSPITAL 1.9432976556852521
DIS_HOME WITH HOME IV PROVIDR 4.872689418581528
DIS_DISCH-TRAN TO PSYCH HOSP 0.07049491317583985
DIS_ICF 1.3744540339681275
DIS_HOSPICE-MEDICAL FACILITY 3.2435782774185453
DIS_HOSPICE-HOME 3.2435782774185453
blood 74.31158446726798
circulatory 43.660640799266126
congenital 0.23731761761887585
digestive 55.89685378149425
endocrine 102.94702193867644
genitourinary 98.08592499857299
infectious 52.458690254604925
injury 42.89823550905321
mental 5.044044422556427
misc 88.77072086494155
muscular 13.03558216220732
neoplasms 1.3345919782462077
nervous 61.87642801853128
pregnancy 1.5875681795379424
prenatal 1.5154186355275479
respiratory 68.55147918391532
skin 29.5219825928465
recovery_day 18.383437134427755
dob 7.080139247278516
```



select the most 18 important features

- 5 dummies from initial categorical varaible: ADM_ELECTIVE, ETH_OTHER/UNKNOWN, ETH_BLACK/AFRICAN,AMERICAN, INS_Medicare, DIS_LONG TERM CARE HOSPITAL.
- 12 numeric variables from bag of word representation of Diagnosis: blood, circulatory, digestive, endocrine ,genitourinary, infectious, injury, misc, muscular, nervous, respiratory, skin.
- recovery_day: number of days passed in hospital during the last hospitalization.

[52]:
```
X1_train=X_train[['ADM_ELECTIVE','ETH_OTHER/UNKNOWN','ETH_BLACK/AFRICAN AMERICAN','INS_Medicare','DIS_LONG TERM CARE HOSPITAL','blood','circulatory','digestive', 'endocrine', 'genitourinary','infectious', 'injury', 'misc','muscular', 'nervous', 'respiratory','skin','recovery_day']]
```

```
X1_test=X_test[['ADM_ELECTIVE','ETH_OTHER/UNKNOWN','ETH_BLACK/AFRICAN␣
↪AMERICAN','INS_Medicare','DIS_LONG TERM CARE␣
↪HOSPITAL','blood','circulatory','digestive', 'endocrine',␣
↪'genitourinary','infectious', 'injury', 'misc', 'muscular','nervous',␣
↪'respiratory','skin','recovery_day']]
```

[53]:
```
X1_train.shape, X1_test.shape
```

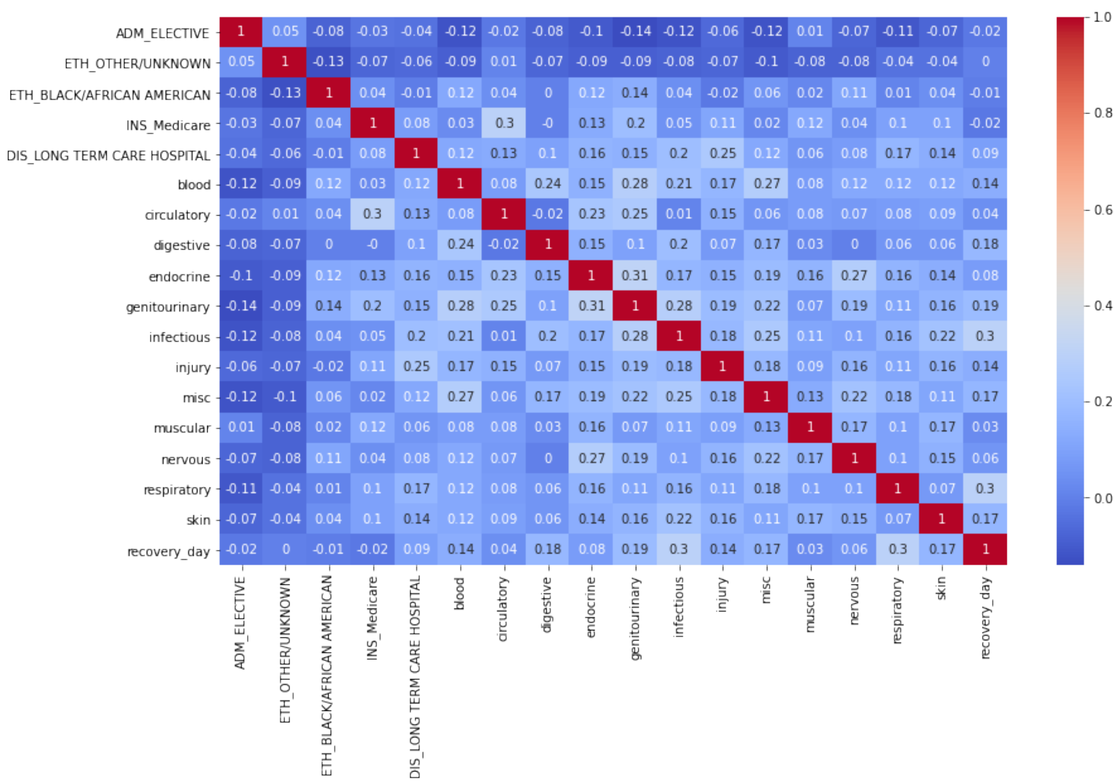
[53]: ((1748, 18), (721, 18))

plot the correlation matrix to check if the features are low correlated.

[54]:
```
X1_train = pd.DataFrame(X1_train)
```

[55]:
```
#plot the correlation of independent variables
# Plot the correlation heatmap
from termcolor import colored as cl
plt.figure(figsize=(14, 8))
corr_matrix = X1_train.corr().round(2)
sns.heatmap(data=corr_matrix,cmap='coolwarm',annot=True)
```

[55]: <AxesSubplot:>



The correlation between the feature seems to be low.

1.8 Neural networks for no text

In this section i will propose 3 different neural networks, than by the evaluation of the metrics I will decide the best one in the conclusion section.

[56]:
```
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.callbacks import EarlyStopping
```

[57]:
```
#import the metcis
from sklearn.metrics import mean_squared_error, r2_score,␣
↪explained_variance_score
```

Option A I create the most possible basic Neural Network and I will see how it perform. There is no neuron between the input and the output block. The aim of this Neural network is to find 19 parameters. The activation function is sigmoid in order to obtain output between 0 and 1 as my scaled days_next_admit.

[58]:
```
modelA = Sequential()
modelA.add(Dense(units=1, input_shape=(18,), activation='sigmoid'))
```

I specify the loss function to use to evaluate a set of weights, I use mean square error since I deal with a regression problem. The optimizer,used to search through different weights for the network, is the efficient stochastic gradient descent algorithm, ADAM an extension to SGD. Finally the metric I would like to collect and report during training is the mean square error.

[59]:
```
# compile the keras model
modelA.compile(loss='mse', optimizer='adam', metrics=[tf.keras.metrics.
↪MeanSquaredError()])
```

[60]:
```
modelA.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 19
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
```

Before train the neural network I set up Earlystopping in order to Stop training when a monitored metric has stopped improving. The Quantity to be monitored is the loss of the validation set. The minimum change in the monitored quantity to be qualify as an improvement is 0.001. After 5 epochs with no improvement the training will be stopped. To print the training epoch on which training was stopped, the "verbose" argument is set to 1. I restore model weights from the epoch with the best value of the monitored quantity. I will use this Earlystopping for all no-text neural networks that I will implement.

[61]:
```
myCallbackNT = EarlyStopping(monitor='loss', min_delta=0.001, patience=5,␣
↪verbose=1, mode='auto', baseline=None, restore_best_weights=True)
```

I put 33% of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.

[62]:
```
historyA=modelA.fit(X1_train, y_train_scaled, validation_split=0.33,␣
↪epochs=100, batch_size=60, callbacks=[myCallbackNT])
```

```
Epoch 1/100
20/20 [==============================] - 1s 48ms/step - loss: 0.4945 -
mean_squared_error: 0.4945 - val_loss: 0.4409 - val_mean_squared_error: 0.4409
Epoch 2/100
20/20 [==============================] - 0s 4ms/step - loss: 0.4645 -
mean_squared_error: 0.4645 - val_loss: 0.4232 - val_mean_squared_error: 0.4232
Epoch 3/100
20/20 [==============================] - 0s 4ms/step - loss: 0.4483 -
mean_squared_error: 0.4483 - val_loss: 0.4000 - val_mean_squared_error: 0.4000
Epoch 4/100
20/20 [==============================] - 0s 4ms/step - loss: 0.4081 -
mean_squared_error: 0.4081 - val_loss: 0.3741 - val_mean_squared_error: 0.3741
Epoch 5/100
20/20 [==============================] - 0s 4ms/step - loss: 0.3812 -
mean_squared_error: 0.3812 - val_loss: 0.3449 - val_mean_squared_error: 0.3449
Epoch 6/100
20/20 [==============================] - 0s 4ms/step - loss: 0.3432 -
mean_squared_error: 0.3432 - val_loss: 0.3141 - val_mean_squared_error: 0.3141
Epoch 7/100
20/20 [==============================] - 0s 4ms/step - loss: 0.3034 -
mean_squared_error: 0.3034 - val_loss: 0.2839 - val_mean_squared_error: 0.2839
Epoch 8/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2683 -
mean_squared_error: 0.2683 - val_loss: 0.2638 - val_mean_squared_error: 0.2638
Epoch 9/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2388 -
mean_squared_error: 0.2388 - val_loss: 0.2568 - val_mean_squared_error: 0.2568
Epoch 10/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2397 -
mean_squared_error: 0.2397 - val_loss: 0.2549 - val_mean_squared_error: 0.2549
```

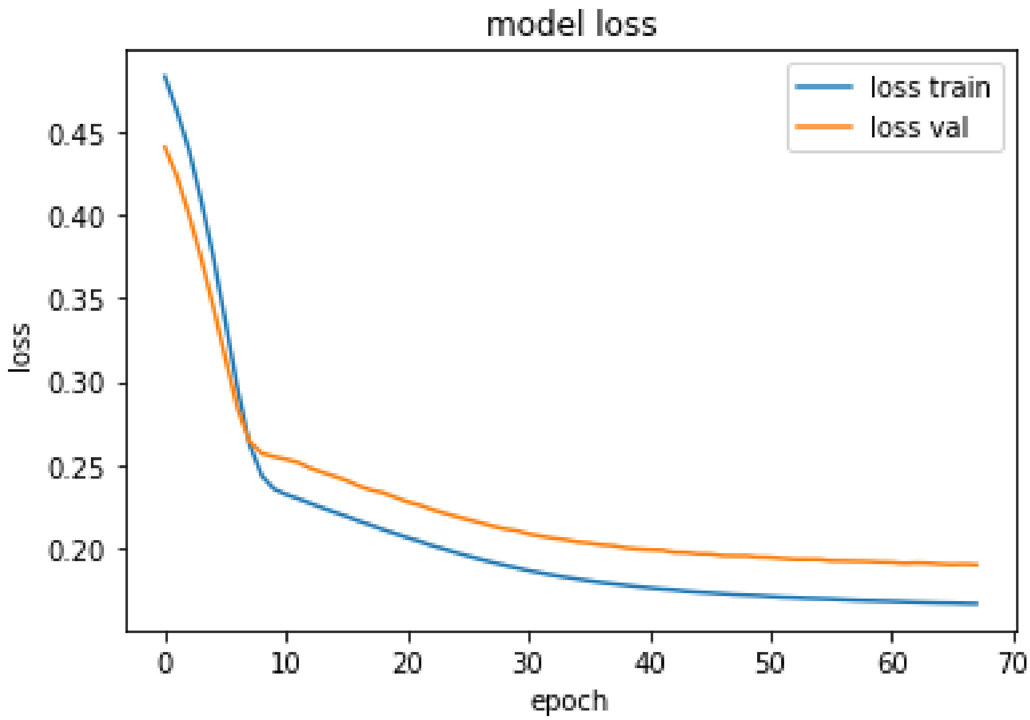
```
Epoch 11/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2436 -
mean_squared_error: 0.2436 - val_loss: 0.2533 - val_mean_squared_error: 0.2533
Epoch 12/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2297 -
mean_squared_error: 0.2297 - val_loss: 0.2515 - val_mean_squared_error: 0.2515
Epoch 13/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2340 -
mean_squared_error: 0.2340 - val_loss: 0.2481 - val_mean_squared_error: 0.2481
Epoch 14/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2237 -
mean_squared_error: 0.2237 - val_loss: 0.2455 - val_mean_squared_error: 0.2455
Epoch 15/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2313 -
mean_squared_error: 0.2313 - val_loss: 0.2432 - val_mean_squared_error: 0.2432
Epoch 16/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2206 -
mean_squared_error: 0.2206 - val_loss: 0.2406 - val_mean_squared_error: 0.2406
Epoch 17/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2088 -
mean_squared_error: 0.2088 - val_loss: 0.2374 - val_mean_squared_error: 0.2374
Epoch 18/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2155 -
mean_squared_error: 0.2155 - val_loss: 0.2350 - val_mean_squared_error: 0.2350
Epoch 19/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2142 -
mean_squared_error: 0.2142 - val_loss: 0.2333 - val_mean_squared_error: 0.2333
Epoch 20/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2144 -
mean_squared_error: 0.2144 - val_loss: 0.2306 - val_mean_squared_error: 0.2306
Epoch 21/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2121 -
mean_squared_error: 0.2121 - val_loss: 0.2278 - val_mean_squared_error: 0.2278
Epoch 22/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2062 -
mean_squared_error: 0.2062 - val_loss: 0.2259 - val_mean_squared_error: 0.2259
Epoch 23/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1955 -
mean_squared_error: 0.1955 - val_loss: 0.2233 - val_mean_squared_error: 0.2233
Epoch 24/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2032 -
mean_squared_error: 0.2032 - val_loss: 0.2211 - val_mean_squared_error: 0.2211
Epoch 25/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2015 -
mean_squared_error: 0.2015 - val_loss: 0.2191 - val_mean_squared_error: 0.2191
Epoch 26/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1917 -
mean_squared_error: 0.1917 - val_loss: 0.2172 - val_mean_squared_error: 0.2172
```

```
Epoch 27/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1956 -
mean_squared_error: 0.1956 - val_loss: 0.2153 - val_mean_squared_error: 0.2153
Epoch 28/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1936 -
mean_squared_error: 0.1936 - val_loss: 0.2133 - val_mean_squared_error: 0.2133
Epoch 29/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1926 -
mean_squared_error: 0.1926 - val_loss: 0.2116 - val_mean_squared_error: 0.2116
Epoch 30/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1887 -
mean_squared_error: 0.1887 - val_loss: 0.2105 - val_mean_squared_error: 0.2105
Epoch 31/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1856 -
mean_squared_error: 0.1856 - val_loss: 0.2086 - val_mean_squared_error: 0.2086
Epoch 32/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1872 -
mean_squared_error: 0.1872 - val_loss: 0.2072 - val_mean_squared_error: 0.2072
Epoch 33/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1820 -
mean_squared_error: 0.1820 - val_loss: 0.2060 - val_mean_squared_error: 0.2060
Epoch 34/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1824 -
mean_squared_error: 0.1824 - val_loss: 0.2051 - val_mean_squared_error: 0.2051
Epoch 35/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1824 -
mean_squared_error: 0.1824 - val_loss: 0.2037 - val_mean_squared_error: 0.2037
Epoch 36/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1769 -
mean_squared_error: 0.1769 - val_loss: 0.2030 - val_mean_squared_error: 0.2030
Epoch 37/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1812 -
mean_squared_error: 0.1812 - val_loss: 0.2020 - val_mean_squared_error: 0.2020
Epoch 38/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1756 -
mean_squared_error: 0.1756 - val_loss: 0.2014 - val_mean_squared_error: 0.2014
Epoch 39/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1837 -
mean_squared_error: 0.1837 - val_loss: 0.2002 - val_mean_squared_error: 0.2002
Epoch 40/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1804 -
mean_squared_error: 0.1804 - val_loss: 0.1994 - val_mean_squared_error: 0.1994
Epoch 41/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1743 -
mean_squared_error: 0.1743 - val_loss: 0.1990 - val_mean_squared_error: 0.1990
Epoch 42/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1755 -
mean_squared_error: 0.1755 - val_loss: 0.1988 - val_mean_squared_error: 0.1988
```

```
Epoch 43/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1842 -
mean_squared_error: 0.1842 - val_loss: 0.1975 - val_mean_squared_error: 0.1975
Epoch 44/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1764 -
mean_squared_error: 0.1764 - val_loss: 0.1973 - val_mean_squared_error: 0.1973
Epoch 45/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1720 -
mean_squared_error: 0.1720 - val_loss: 0.1965 - val_mean_squared_error: 0.1965
Epoch 46/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1737 -
mean_squared_error: 0.1737 - val_loss: 0.1964 - val_mean_squared_error: 0.1964
Epoch 47/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1754 -
mean_squared_error: 0.1754 - val_loss: 0.1954 - val_mean_squared_error: 0.1954
Epoch 48/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1740 -
mean_squared_error: 0.1740 - val_loss: 0.1952 - val_mean_squared_error: 0.1952
Epoch 49/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1708 -
mean_squared_error: 0.1708 - val_loss: 0.1953 - val_mean_squared_error: 0.1953
Epoch 50/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1714 -
mean_squared_error: 0.1714 - val_loss: 0.1945 - val_mean_squared_error: 0.1945
Epoch 51/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1702 -
mean_squared_error: 0.1702 - val_loss: 0.1944 - val_mean_squared_error: 0.1944
Epoch 52/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1732 -
mean_squared_error: 0.1732 - val_loss: 0.1937 - val_mean_squared_error: 0.1937
Epoch 53/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1694 -
mean_squared_error: 0.1694 - val_loss: 0.1932 - val_mean_squared_error: 0.1932
Epoch 54/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1708 -
mean_squared_error: 0.1708 - val_loss: 0.1932 - val_mean_squared_error: 0.1932
Epoch 55/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1649 -
mean_squared_error: 0.1649 - val_loss: 0.1932 - val_mean_squared_error: 0.1932
Epoch 56/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1635 -
mean_squared_error: 0.1635 - val_loss: 0.1923 - val_mean_squared_error: 0.1923
Epoch 57/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1678 -
mean_squared_error: 0.1678 - val_loss: 0.1922 - val_mean_squared_error: 0.1922
Epoch 58/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1667 -
mean_squared_error: 0.1667 - val_loss: 0.1921 - val_mean_squared_error: 0.1921
```

```
Epoch 59/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1761 -
mean_squared_error: 0.1761 - val_loss: 0.1919 - val_mean_squared_error: 0.1919
Epoch 60/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1625 -
mean_squared_error: 0.1625 - val_loss: 0.1916 - val_mean_squared_error: 0.1916
Epoch 61/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1655 -
mean_squared_error: 0.1655 - val_loss: 0.1914 - val_mean_squared_error: 0.1914
Epoch 62/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1659 -
mean_squared_error: 0.1659 - val_loss: 0.1908 - val_mean_squared_error: 0.1908
Epoch 63/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1711 -
mean_squared_error: 0.1711 - val_loss: 0.1911 - val_mean_squared_error: 0.1911
Epoch 64/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1641 -
mean_squared_error: 0.1641 - val_loss: 0.1909 - val_mean_squared_error: 0.1909
Epoch 65/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1654 -
mean_squared_error: 0.1654 - val_loss: 0.1904 - val_mean_squared_error: 0.1904
Epoch 66/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1708 -
mean_squared_error: 0.1708 - val_loss: 0.1901 - val_mean_squared_error: 0.1901
Epoch 67/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1691 -
mean_squared_error: 0.1691 - val_loss: 0.1903 - val_mean_squared_error: 0.1903
Epoch 68/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1687 -
mean_squared_error: 0.1687 - val_loss: 0.1900 - val_mean_squared_error: 0.1900
Restoring model weights from the end of the best epoch.
Epoch 00068: early stopping
```

[63]:
```python
# plot history to check overfitting
# list all data in history

# summarize history for loss
plt.plot(historyA.history['loss'])
plt.plot(historyA.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss train', 'loss val'], loc='best')
plt.show()
```



[64]:
```
# evaluate the keras model train set
modelA.evaluate(X1_train, y_train_scaled)
```

```
55/55 [==============================] - 0s 788us/step - loss: 0.1751 -
mean_squared_error: 0.1751
```

[64]: [0.17513303458690643, 0.17513303458690643]

[65]:
```
#prediction on test set
predictionsA = modelA.predict(X1_test)
```

[66]:
```
#mean square error regression loss
mean_squared_error(y_test_scaled,predictionsA)
```

[66]: 0.18477607763810636

[67]:
```
#R^2 score
r2_score(y_test_scaled,predictionsA)
```

[67]: 0.16830879019030742

[68]:
```
#explained variance
explained_variance_score(y_test_scaled,predictionsA)
```

[68]: 0.19092055685457365

Option B I create a Sequential model and add layers one at a time. Fully connected layers are defined using the Dense class. I use the rectified linear unit activation function referred as ReLU on the first two layers and the Sigmoid function in the output layer. I use a sigmoid on the output layer to ensure our network output is between 0 and 1 as my scaled output variable.

[69]:
```
# define the keras model
modelB = Sequential()
modelB.add(Dense(12, input_dim=18, activation='relu'))
modelB.add(Dense(6, activation='relu'))
modelB.add(Dense(1, activation='sigmoid'))
```

As before I will use mean square error to evaluate a set of weights, as before the optimaizer is ADAM and the main metric I would like to collect is again the mean square error.

[70]:
```
# compile the keras model
modelB.compile(loss='mse', optimizer='adam', metrics=[tf.keras.metrics.
↪MeanSquaredError()])
```

[71]:
```
modelB.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 12)                228
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 78
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7
=================================================================
Total params: 313
Trainable params: 313
Non-trainable params: 0
_________________________________________________________________
```

I use the same Earlystopping previously declared (myCallbackNT) and I put 33% of the training data to be used as validation data.

[72]:
```
# fit model
historyB=modelB.fit(X1_train, y_train_scaled, validation_split=0.33,␣
↪epochs=100, batch_size=60, callbacks=[myCallbackNT])
```

```
Epoch 1/100
20/20 [==============================] - 1s 11ms/step - loss: 0.3200 -
mean_squared_error: 0.3200 - val_loss: 0.3546 - val_mean_squared_error: 0.3546
Epoch 2/100
```

```
20/20 [==============================] - 0s 4ms/step - loss: 0.3051 -
mean_squared_error: 0.3051 - val_loss: 0.3276 - val_mean_squared_error: 0.3276
Epoch 3/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2649 -
mean_squared_error: 0.2649 - val_loss: 0.2894 - val_mean_squared_error: 0.2894
Epoch 4/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2391 -
mean_squared_error: 0.2391 - val_loss: 0.2452 - val_mean_squared_error: 0.2452
Epoch 5/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2245 -
mean_squared_error: 0.2245 - val_loss: 0.2161 - val_mean_squared_error: 0.2161
Epoch 6/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1949 -
mean_squared_error: 0.1949 - val_loss: 0.2092 - val_mean_squared_error: 0.2092
Epoch 7/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1904 -
mean_squared_error: 0.1904 - val_loss: 0.2068 - val_mean_squared_error: 0.2068
Epoch 8/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1866 -
mean_squared_error: 0.1866 - val_loss: 0.2038 - val_mean_squared_error: 0.2038
Epoch 9/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1923 -
mean_squared_error: 0.1923 - val_loss: 0.2025 - val_mean_squared_error: 0.2025
Epoch 10/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1788 -
mean_squared_error: 0.1788 - val_loss: 0.2008 - val_mean_squared_error: 0.2008
Epoch 11/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1759 -
mean_squared_error: 0.1759 - val_loss: 0.1982 - val_mean_squared_error: 0.1982
Epoch 12/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1752 -
mean_squared_error: 0.1752 - val_loss: 0.1979 - val_mean_squared_error: 0.1979
Epoch 13/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1719 -
mean_squared_error: 0.1719 - val_loss: 0.1965 - val_mean_squared_error: 0.1965
Epoch 14/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1707 -
mean_squared_error: 0.1707 - val_loss: 0.1940 - val_mean_squared_error: 0.1940
Epoch 15/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1732 -
mean_squared_error: 0.1732 - val_loss: 0.1929 - val_mean_squared_error: 0.1929
Epoch 16/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1696 -
mean_squared_error: 0.1696 - val_loss: 0.1926 - val_mean_squared_error: 0.1926
Epoch 17/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1662 -
mean_squared_error: 0.1662 - val_loss: 0.1915 - val_mean_squared_error: 0.1915
Epoch 18/100
```

```
20/20 [==============================] - 0s 4ms/step - loss: 0.1659 -
mean_squared_error: 0.1659 - val_loss: 0.1900 - val_mean_squared_error: 0.1900
Epoch 19/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1679 -
mean_squared_error: 0.1679 - val_loss: 0.1905 - val_mean_squared_error: 0.1905
Epoch 20/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1678 -
mean_squared_error: 0.1678 - val_loss: 0.1894 - val_mean_squared_error: 0.1894
Epoch 21/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1616 -
mean_squared_error: 0.1616 - val_loss: 0.1891 - val_mean_squared_error: 0.1891
Epoch 22/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1619 -
mean_squared_error: 0.1619 - val_loss: 0.1880 - val_mean_squared_error: 0.1880
Epoch 23/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1665 -
mean_squared_error: 0.1665 - val_loss: 0.1878 - val_mean_squared_error: 0.1878
Epoch 24/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1585 -
mean_squared_error: 0.1585 - val_loss: 0.1876 - val_mean_squared_error: 0.1876
Epoch 25/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1617 -
mean_squared_error: 0.1617 - val_loss: 0.1864 - val_mean_squared_error: 0.1864
Epoch 26/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1701 -
mean_squared_error: 0.1701 - val_loss: 0.1871 - val_mean_squared_error: 0.1871
Epoch 27/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1608 -
mean_squared_error: 0.1608 - val_loss: 0.1858 - val_mean_squared_error: 0.1858
Epoch 28/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1602 -
mean_squared_error: 0.1602 - val_loss: 0.1863 - val_mean_squared_error: 0.1863
Epoch 29/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1640 -
mean_squared_error: 0.1640 - val_loss: 0.1865 - val_mean_squared_error: 0.1865
Epoch 30/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1676 -
mean_squared_error: 0.1676 - val_loss: 0.1854 - val_mean_squared_error: 0.1854
Epoch 31/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1638 -
mean_squared_error: 0.1638 - val_loss: 0.1850 - val_mean_squared_error: 0.1850
Epoch 32/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1621 -
mean_squared_error: 0.1621 - val_loss: 0.1852 - val_mean_squared_error: 0.1852
Epoch 33/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1563 -
mean_squared_error: 0.1563 - val_loss: 0.1853 - val_mean_squared_error: 0.1853
Epoch 34/100
```

```
20/20 [==============================] - 0s 4ms/step - loss: 0.1547 -
mean_squared_error: 0.1547 - val_loss: 0.1844 - val_mean_squared_error: 0.1844
Epoch 35/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1607 -
mean_squared_error: 0.1607 - val_loss: 0.1843 - val_mean_squared_error: 0.1843
Epoch 36/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1630 -
mean_squared_error: 0.1630 - val_loss: 0.1845 - val_mean_squared_error: 0.1845
Epoch 37/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1564 -
mean_squared_error: 0.1564 - val_loss: 0.1848 - val_mean_squared_error: 0.1848
Epoch 38/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1622 -
mean_squared_error: 0.1622 - val_loss: 0.1839 - val_mean_squared_error: 0.1839
Epoch 39/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1560 -
mean_squared_error: 0.1560 - val_loss: 0.1838 - val_mean_squared_error: 0.1838
Epoch 40/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1580 -
mean_squared_error: 0.1580 - val_loss: 0.1849 - val_mean_squared_error: 0.1849
Epoch 41/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1549 -
mean_squared_error: 0.1549 - val_loss: 0.1842 - val_mean_squared_error: 0.1842
Epoch 42/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1577 -
mean_squared_error: 0.1577 - val_loss: 0.1841 - val_mean_squared_error: 0.1841
Epoch 43/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1488 -
mean_squared_error: 0.1488 - val_loss: 0.1844 - val_mean_squared_error: 0.1844
Epoch 44/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1526 -
mean_squared_error: 0.1526 - val_loss: 0.1836 - val_mean_squared_error: 0.1836
Epoch 45/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1575 -
mean_squared_error: 0.1575 - val_loss: 0.1841 - val_mean_squared_error: 0.1841
Epoch 46/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1587 -
mean_squared_error: 0.1587 - val_loss: 0.1839 - val_mean_squared_error: 0.1839
Epoch 47/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1513 -
mean_squared_error: 0.1513 - val_loss: 0.1847 - val_mean_squared_error: 0.1847
Epoch 48/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1514 -
mean_squared_error: 0.1514 - val_loss: 0.1843 - val_mean_squared_error: 0.1843
Epoch 49/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1535 -
mean_squared_error: 0.1535 - val_loss: 0.1841 - val_mean_squared_error: 0.1841
Epoch 50/100
```

```
20/20 [==============================] - 0s 4ms/step - loss: 0.1521 -
mean_squared_error: 0.1521 - val_loss: 0.1843 - val_mean_squared_error: 0.1843
Epoch 51/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1494 -
mean_squared_error: 0.1494 - val_loss: 0.1844 - val_mean_squared_error: 0.1844
Epoch 52/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1572 -
mean_squared_error: 0.1572 - val_loss: 0.1855 - val_mean_squared_error: 0.1855
Epoch 53/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1498 -
mean_squared_error: 0.1498 - val_loss: 0.1845 - val_mean_squared_error: 0.1845
Epoch 54/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1525 -
mean_squared_error: 0.1525 - val_loss: 0.1847 - val_mean_squared_error: 0.1847
Epoch 55/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1464 -
mean_squared_error: 0.1464 - val_loss: 0.1836 - val_mean_squared_error: 0.1836
Epoch 56/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1485 -
mean_squared_error: 0.1485 - val_loss: 0.1846 - val_mean_squared_error: 0.1846
Epoch 57/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1520 -
mean_squared_error: 0.1520 - val_loss: 0.1839 - val_mean_squared_error: 0.1839
Epoch 58/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1483 -
mean_squared_error: 0.1483 - val_loss: 0.1846 - val_mean_squared_error: 0.1846
Epoch 59/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1553 -
mean_squared_error: 0.1553 - val_loss: 0.1842 - val_mean_squared_error: 0.1842
Epoch 60/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1465 -
mean_squared_error: 0.1465 - val_loss: 0.1853 - val_mean_squared_error: 0.1853
Epoch 61/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1484 -
mean_squared_error: 0.1484 - val_loss: 0.1847 - val_mean_squared_error: 0.1847
Epoch 62/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1483 -
mean_squared_error: 0.1483 - val_loss: 0.1854 - val_mean_squared_error: 0.1854
Epoch 63/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1404 -
mean_squared_error: 0.1404 - val_loss: 0.1850 - val_mean_squared_error: 0.1850
Epoch 64/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1478 -
mean_squared_error: 0.1478 - val_loss: 0.1851 - val_mean_squared_error: 0.1851
Epoch 65/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1469 -
mean_squared_error: 0.1469 - val_loss: 0.1859 - val_mean_squared_error: 0.1859
Epoch 66/100
```

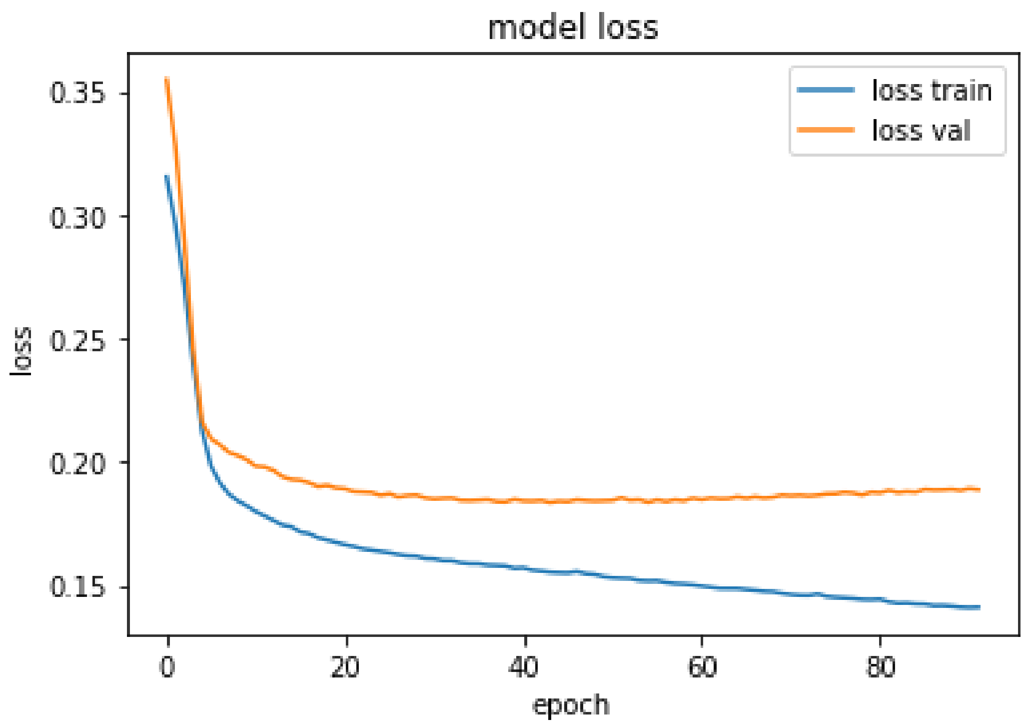
```
20/20 [==============================] - 0s 4ms/step - loss: 0.1458 -
mean_squared_error: 0.1458 - val_loss: 0.1853 - val_mean_squared_error: 0.1853
Epoch 67/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1450 -
mean_squared_error: 0.1450 - val_loss: 0.1859 - val_mean_squared_error: 0.1859
Epoch 68/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1533 -
mean_squared_error: 0.1533 - val_loss: 0.1853 - val_mean_squared_error: 0.1853
Epoch 69/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1428 -
mean_squared_error: 0.1428 - val_loss: 0.1857 - val_mean_squared_error: 0.1857
Epoch 70/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1416 -
mean_squared_error: 0.1416 - val_loss: 0.1865 - val_mean_squared_error: 0.1865
Epoch 71/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1471 -
mean_squared_error: 0.1471 - val_loss: 0.1862 - val_mean_squared_error: 0.1862
Epoch 72/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1402 -
mean_squared_error: 0.1402 - val_loss: 0.1864 - val_mean_squared_error: 0.1864
Epoch 73/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1439 -
mean_squared_error: 0.1439 - val_loss: 0.1862 - val_mean_squared_error: 0.1862
Epoch 74/100
20/20 [==============================] - 0s 5ms/step - loss: 0.1471 -
mean_squared_error: 0.1471 - val_loss: 0.1865 - val_mean_squared_error: 0.1865
Epoch 75/100
20/20 [==============================] - 0s 6ms/step - loss: 0.1442 -
mean_squared_error: 0.1442 - val_loss: 0.1870 - val_mean_squared_error: 0.1870
Epoch 76/100
20/20 [==============================] - 0s 5ms/step - loss: 0.1475 -
mean_squared_error: 0.1475 - val_loss: 0.1871 - val_mean_squared_error: 0.1871
Epoch 77/100
20/20 [==============================] - 0s 6ms/step - loss: 0.1502 -
mean_squared_error: 0.1502 - val_loss: 0.1877 - val_mean_squared_error: 0.1877
Epoch 78/100
20/20 [==============================] - 0s 5ms/step - loss: 0.1552 -
mean_squared_error: 0.1552 - val_loss: 0.1874 - val_mean_squared_error: 0.1874
Epoch 79/100
20/20 [==============================] - 0s 5ms/step - loss: 0.1481 -
mean_squared_error: 0.1481 - val_loss: 0.1866 - val_mean_squared_error: 0.1866
Epoch 80/100
20/20 [==============================] - 0s 5ms/step - loss: 0.1455 -
mean_squared_error: 0.1455 - val_loss: 0.1877 - val_mean_squared_error: 0.1877
Epoch 81/100
20/20 [==============================] - 0s 5ms/step - loss: 0.1423 -
mean_squared_error: 0.1423 - val_loss: 0.1875 - val_mean_squared_error: 0.1875
Epoch 82/100
```

```
20/20 [==============================] - 0s 4ms/step - loss: 0.1452 -
mean_squared_error: 0.1452 - val_loss: 0.1885 - val_mean_squared_error: 0.1885
Epoch 83/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1435 -
mean_squared_error: 0.1435 - val_loss: 0.1876 - val_mean_squared_error: 0.1876
Epoch 84/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1361 -
mean_squared_error: 0.1361 - val_loss: 0.1880 - val_mean_squared_error: 0.1880
Epoch 85/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1518 -
mean_squared_error: 0.1518 - val_loss: 0.1877 - val_mean_squared_error: 0.1877
Epoch 86/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1526 -
mean_squared_error: 0.1526 - val_loss: 0.1889 - val_mean_squared_error: 0.1889
Epoch 87/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1398 -
mean_squared_error: 0.1398 - val_loss: 0.1886 - val_mean_squared_error: 0.1886
Epoch 88/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1399 -
mean_squared_error: 0.1399 - val_loss: 0.1887 - val_mean_squared_error: 0.1887
Epoch 89/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1477 -
mean_squared_error: 0.1477 - val_loss: 0.1889 - val_mean_squared_error: 0.1889
Epoch 90/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1422 -
mean_squared_error: 0.1422 - val_loss: 0.1884 - val_mean_squared_error: 0.1884
Epoch 91/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1446 -
mean_squared_error: 0.1446 - val_loss: 0.1892 - val_mean_squared_error: 0.1892
Epoch 92/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1347 -
mean_squared_error: 0.1347 - val_loss: 0.1887 - val_mean_squared_error: 0.1887
Restoring model weights from the end of the best epoch.
Epoch 00092: early stopping
```

[73]:

```
import matplotlib.pyplot as plt
%matplotlib inline

# plot history
# list all data in history

# summarize history for loss
plt.plot(historyB.history['loss'])
plt.plot(historyB.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
```

```
plt.legend(['loss train', 'loss val'], loc='best')
plt.show()

# summarize history for loss
#plt.plot(history.history['acc'])
#plt.plot(history.history['val_acc'])
#plt.title('model acc')
#plt.ylabel('acc')
#plt.xlabel('epoch')
#plt.legend(['acc train', 'acc val'], loc='best')
#plt.show()
```



[74]:
```
# evaluate the keras model train set
modelB.evaluate(X1_train, y_train_scaled)
```

```
55/55 [==============================] - 0s 805us/step - loss: 0.1570 -
mean_squared_error: 0.1570
```

[74]: [0.15695761144161224, 0.15695761144161224]

[75]:
```
#store the prediction on the test set
predictionsB = modelB.predict(X1_test)
```

[76]:
```
#Mean squared error regression loss
mean_squared_error(y_test_scaled, predictionsB)
```

[76]: 0.18787478367791366

[77]:
```
#R^2 (coefficient of determination) regression score function
r2_score(y_test_scaled,predictionsB)
```

[77]: 0.1543612781095528

[78]:
```
#Explained variance regression score function.
explained_variance_score(y_test_scaled,predictionsB)
```

[78]: 0.16291598324389778

Option C As in option B, I create a Sequential model and add layers one at a time. I use one fully connected layers between the input and the output. Some research suggested the number of neural nodes in hidden layers to be between 2/3 to 2 times of the size of the input layer. Since I have 18 features as input I take 11 hidden layers. As before I use the rectified linear unit activation function referred to as ReLU on the first layers and the Sigmoid function in the output layer.

[79]:
```
#define the keras model
modelC = Sequential()
modelC.add(Dense(11, input_dim=18, activation='relu'))
modelC.add(Dense(1, activation='sigmoid'))
```

As before I will use mean square error to evaluate a set of weights, as before the optimaizer is ADAM and the main metric I would like to collect is again the mean square error.

[80]:
```
# compile the keras model
modelC.compile(loss='mse', optimizer='adam', metrics=[tf.keras.metrics.
 ↪MeanSquaredError()])
```

[81]:
```
modelC.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 11)                209
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 12
=================================================================
Total params: 221
Trainable params: 221
Non-trainable params: 0
_________________________________________________________________
```

I use the same Earlystopping previously declared (myCallbackNT) and I put 33% of the training data to be used as validation data.

[82]:
```
historyC=modelC.fit(X1_train, y_train_scaled, validation_split=0.33,␣
↪epochs=100, batch_size=60, callbacks=[myCallbackNT])
```

```
Epoch 1/100
20/20 [==============================] - 1s 11ms/step - loss: 0.2923 -
mean_squared_error: 0.2923 - val_loss: 0.2426 - val_mean_squared_error: 0.2426
Epoch 2/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2306 -
mean_squared_error: 0.2306 - val_loss: 0.2311 - val_mean_squared_error: 0.2311
Epoch 3/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2216 -
mean_squared_error: 0.2216 - val_loss: 0.2249 - val_mean_squared_error: 0.2249
Epoch 4/100
20/20 [==============================] - 0s 4ms/step - loss: 0.2120 -
mean_squared_error: 0.2120 - val_loss: 0.2184 - val_mean_squared_error: 0.2184
Epoch 5/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1976 -
mean_squared_error: 0.1976 - val_loss: 0.2139 - val_mean_squared_error: 0.2139
Epoch 6/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1900 -
mean_squared_error: 0.1900 - val_loss: 0.2090 - val_mean_squared_error: 0.2090
Epoch 7/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1904 -
mean_squared_error: 0.1904 - val_loss: 0.2052 - val_mean_squared_error: 0.2052
Epoch 8/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1809 -
mean_squared_error: 0.1809 - val_loss: 0.2033 - val_mean_squared_error: 0.2033
Epoch 9/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1893 -
mean_squared_error: 0.1893 - val_loss: 0.2012 - val_mean_squared_error: 0.2012
Epoch 10/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1813 -
mean_squared_error: 0.1813 - val_loss: 0.1997 - val_mean_squared_error: 0.1997
Epoch 11/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1750 -
mean_squared_error: 0.1750 - val_loss: 0.1971 - val_mean_squared_error: 0.1971
Epoch 12/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1752 -
mean_squared_error: 0.1752 - val_loss: 0.1966 - val_mean_squared_error: 0.1966
Epoch 13/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1849 -
mean_squared_error: 0.1849 - val_loss: 0.1953 - val_mean_squared_error: 0.1953
Epoch 14/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1762 -
mean_squared_error: 0.1762 - val_loss: 0.1939 - val_mean_squared_error: 0.1939
```

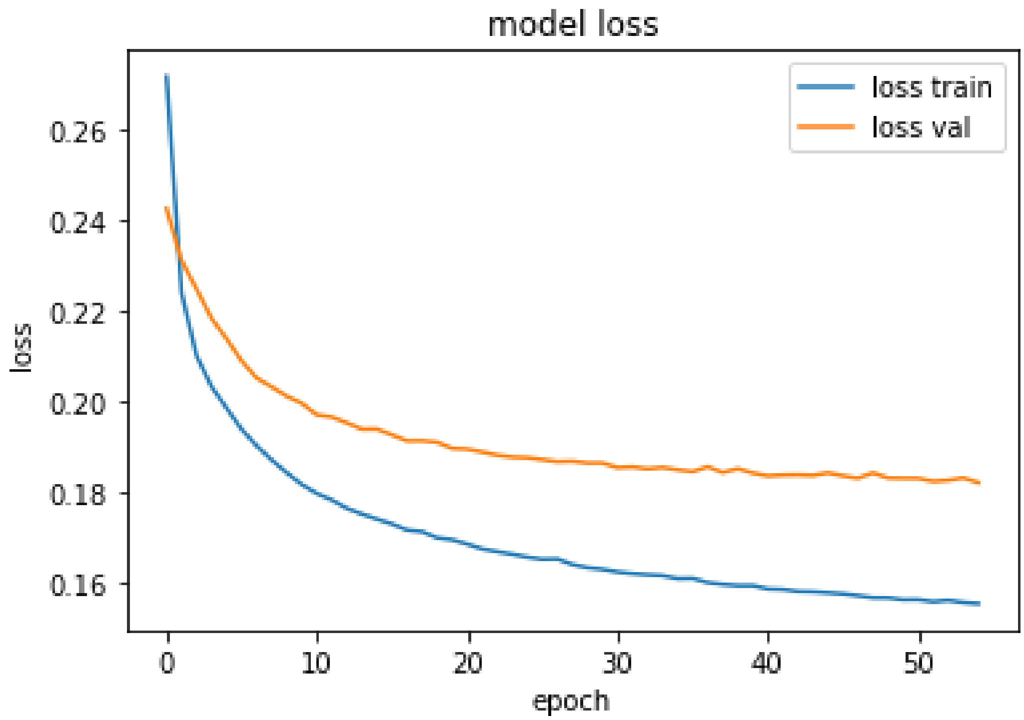
```
Epoch 15/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1739 -
mean_squared_error: 0.1739 - val_loss: 0.1939 - val_mean_squared_error: 0.1939
Epoch 16/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1811 -
mean_squared_error: 0.1811 - val_loss: 0.1927 - val_mean_squared_error: 0.1927
Epoch 17/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1706 -
mean_squared_error: 0.1706 - val_loss: 0.1913 - val_mean_squared_error: 0.1913
Epoch 18/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1718 -
mean_squared_error: 0.1718 - val_loss: 0.1913 - val_mean_squared_error: 0.1913
Epoch 19/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1677 -
mean_squared_error: 0.1677 - val_loss: 0.1910 - val_mean_squared_error: 0.1910
Epoch 20/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1684 -
mean_squared_error: 0.1684 - val_loss: 0.1897 - val_mean_squared_error: 0.1897
Epoch 21/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1648 -
mean_squared_error: 0.1648 - val_loss: 0.1895 - val_mean_squared_error: 0.1895
Epoch 22/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1643 -
mean_squared_error: 0.1643 - val_loss: 0.1889 - val_mean_squared_error: 0.1889
Epoch 23/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1659 -
mean_squared_error: 0.1659 - val_loss: 0.1882 - val_mean_squared_error: 0.1882
Epoch 24/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1662 -
mean_squared_error: 0.1662 - val_loss: 0.1878 - val_mean_squared_error: 0.1878
Epoch 25/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1653 -
mean_squared_error: 0.1653 - val_loss: 0.1877 - val_mean_squared_error: 0.1877
Epoch 26/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1567 -
mean_squared_error: 0.1567 - val_loss: 0.1872 - val_mean_squared_error: 0.1872
Epoch 27/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1714 -
mean_squared_error: 0.1714 - val_loss: 0.1868 - val_mean_squared_error: 0.1868
Epoch 28/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1607 -
mean_squared_error: 0.1607 - val_loss: 0.1869 - val_mean_squared_error: 0.1869
Epoch 29/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1610 -
mean_squared_error: 0.1610 - val_loss: 0.1865 - val_mean_squared_error: 0.1865
Epoch 30/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1622 -
mean_squared_error: 0.1622 - val_loss: 0.1865 - val_mean_squared_error: 0.1865
```

```
Epoch 31/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1560 -
mean_squared_error: 0.1560 - val_loss: 0.1855 - val_mean_squared_error: 0.1855
Epoch 32/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1619 -
mean_squared_error: 0.1619 - val_loss: 0.1856 - val_mean_squared_error: 0.1856
Epoch 33/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1583 -
mean_squared_error: 0.1583 - val_loss: 0.1852 - val_mean_squared_error: 0.1852
Epoch 34/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1651 -
mean_squared_error: 0.1651 - val_loss: 0.1855 - val_mean_squared_error: 0.1855
Epoch 35/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1594 -
mean_squared_error: 0.1594 - val_loss: 0.1850 - val_mean_squared_error: 0.1850
Epoch 36/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1595 -
mean_squared_error: 0.1595 - val_loss: 0.1846 - val_mean_squared_error: 0.1846
Epoch 37/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1635 -
mean_squared_error: 0.1635 - val_loss: 0.1856 - val_mean_squared_error: 0.1856
Epoch 38/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1648 -
mean_squared_error: 0.1648 - val_loss: 0.1844 - val_mean_squared_error: 0.1844
Epoch 39/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1703 -
mean_squared_error: 0.1703 - val_loss: 0.1852 - val_mean_squared_error: 0.1852
Epoch 40/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1610 -
mean_squared_error: 0.1610 - val_loss: 0.1843 - val_mean_squared_error: 0.1843
Epoch 41/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1626 -
mean_squared_error: 0.1626 - val_loss: 0.1836 - val_mean_squared_error: 0.1836
Epoch 42/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1600 -
mean_squared_error: 0.1600 - val_loss: 0.1838 - val_mean_squared_error: 0.1838
Epoch 43/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1595 -
mean_squared_error: 0.1595 - val_loss: 0.1839 - val_mean_squared_error: 0.1839
Epoch 44/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1563 -
mean_squared_error: 0.1563 - val_loss: 0.1837 - val_mean_squared_error: 0.1837
Epoch 45/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1582 -
mean_squared_error: 0.1582 - val_loss: 0.1843 - val_mean_squared_error: 0.1843
Epoch 46/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1614 -
mean_squared_error: 0.1614 - val_loss: 0.1836 - val_mean_squared_error: 0.1836
```

```
Epoch 47/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1522 -
mean_squared_error: 0.1522 - val_loss: 0.1831 - val_mean_squared_error: 0.1831
Epoch 48/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1556 -
mean_squared_error: 0.1556 - val_loss: 0.1843 - val_mean_squared_error: 0.1843
Epoch 49/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1626 -
mean_squared_error: 0.1626 - val_loss: 0.1832 - val_mean_squared_error: 0.1832
Epoch 50/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1555 -
mean_squared_error: 0.1555 - val_loss: 0.1831 - val_mean_squared_error: 0.1831
Epoch 51/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1464 -
mean_squared_error: 0.1464 - val_loss: 0.1831 - val_mean_squared_error: 0.1831
Epoch 52/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1547 -
mean_squared_error: 0.1547 - val_loss: 0.1824 - val_mean_squared_error: 0.1824
Epoch 53/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1577 -
mean_squared_error: 0.1577 - val_loss: 0.1827 - val_mean_squared_error: 0.1827
Epoch 54/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1607 -
mean_squared_error: 0.1607 - val_loss: 0.1832 - val_mean_squared_error: 0.1832
Epoch 55/100
20/20 [==============================] - 0s 4ms/step - loss: 0.1535 -
mean_squared_error: 0.1535 - val_loss: 0.1821 - val_mean_squared_error: 0.1821
Restoring model weights from the end of the best epoch.
Epoch 00055: early stopping
```

[83]:

```
# plot history to check overfitting
# list all data in history

# summarize history for loss
plt.plot(historyC.history['loss'])
plt.plot(historyC.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss train', 'loss val'], loc='best')
plt.show()
```



[84]:
```
# evaluate the keras model train set
modelC.evaluate(X1_train, y_train_scaled)
```

```
55/55 [==============================] - 0s 823us/step - loss: 0.1649 -
mean_squared_error: 0.1649
```

[84]: [0.16489346325397491, 0.16489346325397491]

[85]:
```
#prediction on the test set
predictionsC = modelC.predict(X1_test)
```

[86]:
```
#mean square error regression loss
mean_squared_error(y_test_scaled,predictionsC)
```

[86]: 0.17832514416861525

[87]:
```
#R^2 score
r2_score(y_test_scaled,predictionsC)
```

[87]: 0.19734493345204984

[88]:
```
#explained variance score
explained_variance_score(y_test_scaled,predictionsC)
```

[88]: 0.20987429737932484

[89]:
```
#prediction in days
predictionsC_notscaled=scaler.inverse_transform(predictionsC)
#mse on Number of days
mse_notext=mean_squared_error(y_test,predictionsC_notscaled)
```

1.9 Neural network for no text and text info

Before proceeding with the implementation of the neural network with no-text and text info, I will treat normalized doctor's reports.

Each normalized clinical report is a variable sequence of words and the information inside each report must be used to predict the number of days before the next re-hospitalization.

[90]:
```
#selection of test from train and test set
test=df_test1['text']
train=df_train1['text']
```

I look at the train set in order to decide the size of interest (How many word I would like to model).

[91]:
```
# top 500 word frequencies
top_N = 600
a = train.str.cat(sep=' ')
words = nltk.tokenize.word_tokenize(a)
word_dist = nltk.FreqDist(words)
rslt = pd.DataFrame(word_dist.most_common(top_N),
                    columns=['Word', 'Frequency'])
```

[92]:
```
pd.set_option("display.max_rows", None, "display.max_columns", None)
```

[93]:
```
print(rslt)
```

```
               Word  Frequency
0                wa      40352
1                po      17905
2               one      11050
3                hi       9893
4              last       7694
5              left       6699
6                pt       5625
7             right       5285
8            hospit       4796
9              pain       4505
10              thi       4477
11         hospital       3686
12               ha       3678
13             need       3446
```

14	ct	3283
15	continu	3116
16	normal	2962
17	first	2920
18	refil	2514
19	disp	2390
20	qh	2330
21	chest	2309
22	capsul	2267
23	pleas	2205
24	start	2065
25	everi	2040
26	sp	1952
27	given	1899
28	arteri	1853
29	statu	1756
30	valv	1693
31	releas	1687
32	unit	1677
33	hct	1601
34	home	1562
35	neg	1552
36	dr	1533
37	show	1527
38	aortic	1520
39	onc	1519
40	stitl	1518
41	seen	1467
42	diseas	1461
43	qd	1408
44	ventricular	1359
45	tube	1330
46	un	1307
47	chang	1296
48	bleed	1288
49	bid	1273
50	week	1265
51	well	1233
52	pulmonari	1215
53	stabl	1210
54	bilater	1182
55	renal	1170
56	two	1166
57	pressur	1118
58	cultur	1101
59	effus	1091
60	cours	1078
61	iv	1074

62	wbc	1068
63	cardiac	1066
64	prior	1066
65	l	1058
66	urin	1048
67	initi	1025
68	q	1022
69	fluid	997
70	inr	995
71	improv	992
72	postop	990
73	increas	989
74	like	987
75	plt	965
76	prn	960
77	heart	945
78	sever	904
79	take	904
80	failur	900
81	lung	893
82	place	893
83	mild	879
84	r	876
85	acut	852
86	transfer	849
87	chronic	845
88	hd	840
89	respiratori	838
90	coronari	827
91	k	826
92	due	813
93	care	809
94	mitral	803
95	locat	796
96	small	792
97	deni	789
98	ani	781
99	known	758
100	hypertens	744
101	hr	744
102	pneumonia	733
103	fractur	727
104	abdomin	721
105	dose	719
106	rbc	700
107	insulin	694
108	hgb	687
109	na	679

110	breath	671
111	liver	670
112	post	664
113	moder	651
114	mouth	646
115	coumadin	646
116	fever	646
117	dure	631
118	final	627
119	edema	611
120	glucose	611
121	cl	610
122	systol	602
123	surgeri	601
124	creat	596
125	pleural	594
126	line	591
127	evid	587
128	stent	584
129	without	584
130	mcv	584
131	atrial	583
132	rdw	581
133	mch	580
134	mchc	580
135	infect	574
136	cm	564
137	mass	562
138	urean	549
139	hco	549
140	recent	540
141	remain	539
142	requir	538
143	intub	534
144	lower	529
145	head	522
146	tid	520
147	hypotens	519
148	lasix	511
149	angap	503
150	delay	497
151	c	480
152	reveal	478
153	ec	470
154	mental	466
155	bowel	460
156	graft	457
157	twice	451

158	year	450
159	lesion	450
160	wall	450
161	doctor	444
162	use	443
163	bp	442
164	prednison	441
165	vancomycin	437
166	cathet	431
167	elev	424
168	studi	423
169	regurgit	419
170	dialysi	412
171	extrem	411
172	lastnam	410
173	wound	409
174	stenosi	408
175	lobe	406
176	secondari	403
177	inhal	403
178	gi	402
179	copd	401
180	impress	394
181	seizur	387
182	calcium	386
183	transplant	382
184	ed	382
185	rate	378
186	appoint	375
187	diabet	373
188	receiv	373
189	mr	370
190	treat	367
191	vein	366
192	posit	363
193	exam	362
194	ho	357
195	metoprolol	356
196	alcohol	351
197	cell	347
198	leaflet	346
199	howev	345
200	new	339
201	may	339
202	found	338
203	hemorrhag	333
204	clinic	331
205	three	331

206	mildli	331
207	patch	329
208	contrast	328
209	dilat	323
210	month	321
211	within	320
212	pod	319
213	heparin	306
214	result	306
215	examin	306
216	appear	306
217	trach	304
218	w	304
219	baselin	301
220	hepat	297
221	ni	296
222	aspir	294
223	recommend	292
224	concern	289
225	remov	289
226	chf	286
227	size	283
228	evalu	282
229	decreas	282
230	level	279
231	fibril	279
232	ulcer	278
233	function	278
234	pericardi	276
235	osh	275
236	short	273
237	kidney	272
238	outpati	271
239	intact	270
240	aorta	269
241	back	267
242	feed	267
243	low	265
244	stool	263
245	cxr	261
246	placement	259
247	creatinin	259
248	antibiot	259
249	colon	257
250	find	256
251	stop	255
252	mm	254
253	hematoma	252

254	hematocrit	251
255	incis	251
256	clear	250
257	ascit	250
258	base	250
259	provid	248
260	mri	248
261	diarrhea	247
262	cancer	244
263	lab	241
264	procedur	239
265	cad	237
266	complet	237
267	upper	235
268	cirrhosi	235
269	total	235
270	pancreat	234
271	gram	233
272	set	233
273	brain	233
274	perform	232
275	solut	230
276	oxygen	230
277	steroid	230
278	sodium	229
279	consult	224
280	possibl	224
281	rehab	223
282	number	223
283	drain	222
284	sustain	219
285	obstruct	219
286	imag	216
287	subdur	216
288	demonstr	213
289	neurolog	211
290	infarct	211
291	onli	211
292	drop	210
293	famili	209
294	multipl	209
295	floor	207
296	varic	207
297	nausea	205
298	carotid	203
299	control	202
300	headach	201
301	catheter	196

302	admit	195
303	restart	195
304	ptt	194
305	identifi	192
306	rhythm	191
307	symptom	191
308	lf	191
309	biopsi	190
310	abdomen	190
311	etoh	189
312	larg	189
313	hemodialysi	186
314	episod	186
315	ward	186
316	amiodaron	185
317	cough	184
318	micu	182
319	hip	181
320	fistula	180
321	worsen	180
322	mgdl	179
323	ph	178
324	site	178
325	sinc	175
326	oper	175
327	icu	174
328	tachycardia	172
329	leg	172
330	thicken	172
331	esrd	170
332	monthdayyear	170
333	ventricl	169
334	felt	168
335	anemia	167
336	cath	166
337	signific	164
338	picc	163
339	lad	163
340	distal	162
341	abus	162
342	intraven	161
343	abscess	160
344	develop	160
345	tricuspid	160
346	instruct	158
347	swallow	157
348	import	157
349	sbp	156

350	collect	156
351	consist	155
352	frontal	154
353	sensit	153
354	phos	151
355	held	151
356	withdraw	151
357	toler	150
358	bipap	149
359	lactulos	148
360	count	148
361	tissu	148
362	rang	147
363	activ	147
364	age	144
365	vomit	143
366	icd	143
367	warfarin	142
368	fall	142
369	metastat	141
370	pco	140
371	ctropnt	140
372	aspirin	139
373	egd	138
374	esophag	137
375	replac	136
376	primari	136
377	stroke	136
378	surgic	135
379	node	135
380	growth	135
381	echo	135
382	physic	134
383	foot	134
384	hernia	131
385	suggest	130
386	portal	130
387	bypass	129
388	abl	129
389	neck	129
390	uti	128
391	pcp	126
392	sugar	126
393	spine	124
394	proxim	124
395	aneurysm	124
396	encephalopathi	124
397	intens	123

398	skin	123
399	namei	122
400	tumor	122
401	includ	122
402	unchang	122
403	foley	121
404	ck	121
405	sepsi	121
406	b	121
407	drainag	121
408	repeat	121
409	peg	121
410	pe	120
411	type	120
412	wean	120
413	prescrib	119
414	meropenem	119
415	depress	119
416	vs	119
417	mcg	119
418	drip	118
419	dissect	118
420	sinu	117
421	posterior	117
422	ekg	116
423	vascular	115
424	bronchoscopi	115
425	monthday	115
426	lisinopril	115
427	dyspnea	115
428	xs	114
429	rca	113
430	regimen	113
431	abnorm	113
432	co	113
433	transfus	112
434	test	112
435	area	111
436	ef	111
437	colonoscopi	111
438	scan	111
439	tracheostomi	111
440	medicin	111
441	weak	110
442	digoxin	109
443	htn	109
444	exacerb	108
445	resect	108

446	lymph	108
447	femor	108
448	congest	107
449	repair	107
450	atrium	107
451	sleep	107
452	ms	107
453	sputum	107
454	injuri	106
455	arrest	105
456	vt	105
457	cabg	105
458	constip	104
459	pseudomona	104
460	obes	103
461	diff	103
462	underw	102
463	ago	101
464	thought	101
465	therapi	101
466	oral	101
467	upon	100
468	reason	100
469	veget	99
470	lymphoma	99
471	room	99
472	qday	98
473	sign	98
474	gtt	98
475	mrsa	97
476	flow	97
477	coliti	97
478	stay	97
479	afib	97
480	manag	96
481	resolv	96
482	ventil	96
483	opac	95
484	taper	95
485	check	95
486	distress	94
487	abov	94
488	tab	94
489	extend	94
490	high	93
491	nl	92
492	pacemak	91
493	resist	91

494	arm	91
495	air	91
496	klebsiella	90
497	biliari	90
498	ostomi	89
499	gallbladd	89
500	cervic	89
501	asthma	89
502	sat	89
503	tip	89
504	interv	88
505	throughout	88
506	drink	88
507	ckmb	87
508	platelet	87
509	methadon	87
510	typeart	85
511	recurr	85
512	pneumothorax	84
513	side	84
514	subcutan	84
515	diastol	84
516	treatment	84
517	field	83
518	state	83
519	patent	83
520	spinal	82
521	malign	82
522	confus	82
523	current	82
524	gvhd	82
525	labetalol	81
526	sob	81
527	addit	80
528	ceftriaxon	80
529	anterior	80
530	descend	80
531	approxim	79
532	nodul	79
533	work	79
534	sternal	79
535	lead	78
536	vitamin	78
537	breast	78
538	weight	78
539	lovenox	78
540	tpn	77
541	give	77

542	diet	77
543	eye	77
544	soft	77
545	doppler	77
546	clot	77
547	doe	77
548	sourc	77
549	puff	76
550	issu	76
551	four	76
552	rash	76
553	digit	76
554	subsequ	76
555	clonidin	75
556	mechan	75
557	linezolid	75
558	hiv	75
559	cc	75
560	bedtim	75
561	agit	75
562	alter	74
563	cocain	74
564	diffus	74
565	limit	74
566	free	74
567	dka	74
568	gtube	74
569	extub	74
570	ercp	74
571	syndrom	73
572	obtain	73
573	eeg	73
574	inferior	73
575	bone	72
576	focal	72
577	nurs	72
578	anticoagul	72
579	plavix	72
580	caus	72
581	bacteremia	72
582	wheez	72
583	pong	71
584	phone	71
585	monitor	71
586	flagyl	71
587	facial	70
588	titl	70
589	hand	69

590	mid	69
591	v	69
592	becaus	69
593	lactate	69
594	wife	69
595	pna	69
596	diagnosi	68
597	st	68
598	cord	68
599	emerg	68

Taking into account the most 500 frequent words seems resonable, the occurence of the last words is near 100 times.

I decided to treat text info with Bag of Word and NLP.

I will limit the total number of words that I am interested in modeling to the 500 most frequent words.

[94]:
```
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Embedding,Input, concatenate
from tensorflow.keras.preprocessing import sequence
```

[95]:
```
from tensorflow.keras.preprocessing.text import Tokenizer

from nltk.corpus import stopwords
#nltk.download('stopwords')


FILTERS='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'
VOCABULARY_SIZE = 500 # Max the vocabulary size look at the frequancies below␣
↪to decide it

# create the tokenizer
tokenizer = Tokenizer(num_words=VOCABULARY_SIZE,
              filters=FILTERS,
              split=' ',
              lower=True,
              oov_token="_UNK_")

# fit the tokenizer on the documents
tokenizer.fit_on_texts(train)
print('Found %s unique tokens.' % len(tokenizer.word_index))

# encode documents
X_train_enc = tokenizer.texts_to_sequences(train)
X_test_enc = tokenizer.texts_to_sequences(test)
```

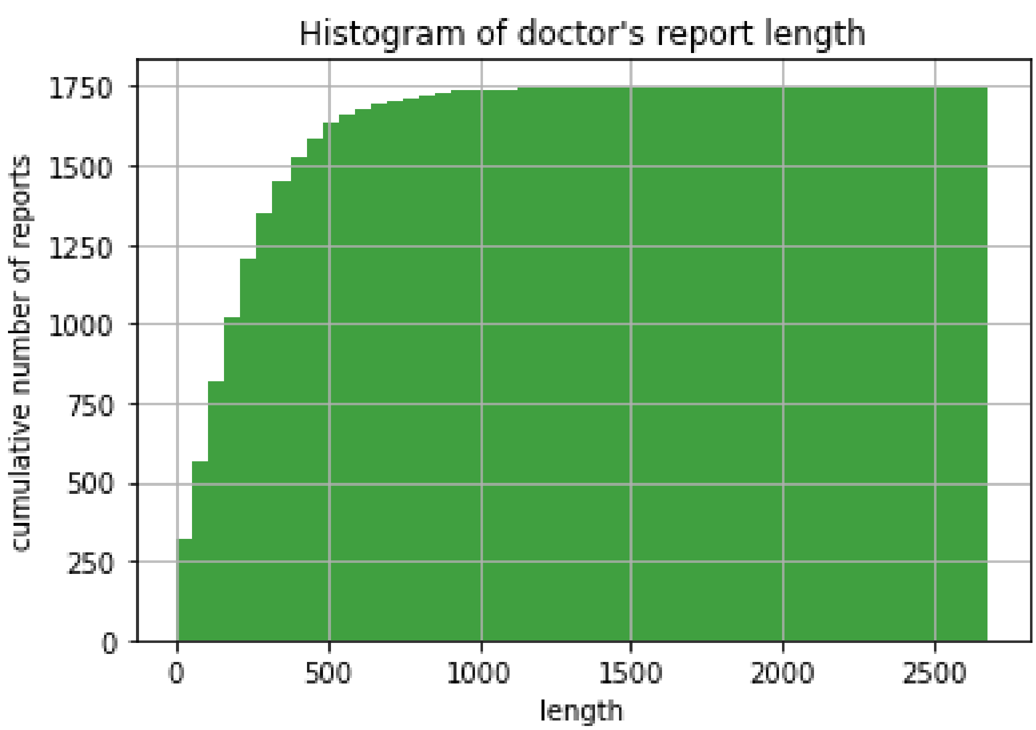
```
Found 2548 unique tokens.
```

Since each tokenized report have different length, I will decide a cut off point, to standardize the length of each report. To do that I will look at the cumulative distribution of the tokenized reports.

[96]:
```
#text size on the train set

length = []
for review in X_train_enc:
    length += [len(review)]

max_length = max(length)
max_length
```

[96]: 2679

[97]:
```
#the histogram of the data
plt.hist(length, 50, density=False, cumulative=True, facecolor='g', alpha=0.75)

plt.xlabel('length')
plt.ylabel('cumulative number of reports')
plt.title("Histogram of doctor's report length")
plt.grid(True)
plt.show()
```



Almost all tokenized clinical reports have less than 1000 words. I decide to truncate at 750 in order to reduce the loss of information and speed up the time to train the neural network. The sequence length (number of words) in each reports varies, so I will constrain each report to be 750 words, truncating long report and pad the shorter reports with zero values.

[98]:
```
# keep the top n words, zero the rest
top_words = 500
# truncate and pad input sequences
max_report_length = 750
X2_train = sequence.pad_sequences(X_train_enc, maxlen=max_report_length)
X2_test = sequence.pad_sequences(X_test_enc, maxlen=max_report_length)
```

1.9.1 The network

In this neural network I try to combine numeric features (numeric_input) and text information.

I use a LSTM model and combine its output with the numeric features. Therefore I define two input layers and treat them in separate models (nlp_input and numeric_input). The NLP data goes through the embedding transformation and the LSTM layer. The first layer is the Embedded layer that uses 16 length vectors to represent each word. The next layer is the LSTM layer with 80 memory units (smart neurons). The numeric_input is just used as it is, so I can just concatenate it with the lstm output (nlp_out). This combined vector is now passed in the finally sigmoid dense layer.

[99]:
```
#create the model
embedding_vecor_length=16
nlp_input = Input(shape=(max_report_length,), name='nlp_input') #each report in␣
 ↪composed by 750 token (pad sequence previously compute)
numeric_input = Input(shape=(18,), name='numeric_input') #18 features␣
 ↪previously selected
emb = Embedding(top_words, embedding_vecor_length,␣
 ↪input_length=max_report_length)(nlp_input)
nlp_out =(LSTM(80))(emb)
x = concatenate([nlp_out, numeric_input])
x = Dense(1, activation='sigmoid')(x)
modelD = Model(inputs=[nlp_input , numeric_input], outputs=[x])
```

[100]:
```
modelD.summary()
```

```
Model: "model"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
nlp_input (InputLayer)          [(None, 750)]        0

________________________________________________________________________________
__________________
embedding (Embedding)           (None, 750, 16)      8000        nlp_input[0][0]
```

```
__________________________________________________________________________________
__________________
lstm (LSTM)                     (None, 80)           31040       embedding[0][0]
__________________________________________________________________________________
__________________
numeric_input (InputLayer)      [(None, 18)]         0
__________________________________________________________________________________
__________________
concatenate (Concatenate)       (None, 98)           0           lstm[0][0]
numeric_input[0][0]
__________________________________________________________________________________
__________________
dense_6 (Dense)                 (None, 1)            99
concatenate[0][0]
==================================================================================
==================
Total params: 39,139
Trainable params: 39,139
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

Although I will train the neural network for few epoch I set up Earlystopping. The Quantity to be monitored is again the mean square error of the validation set. The minimum change in the monitored quantity to be qualify as an improvement is 0.001. After 1 epochs with no improvement the training will be stopped.

[101]:
```
ourCallbackD = EarlyStopping(monitor='loss', min_delta=0.001, patience=1,␣
↪verbose=1, mode='auto', baseline=None, restore_best_weights=True)
```

As before I will use mean square error to evaluate a set of weights, as before the optimaizer is ADAM and the main metric I would like to collect is again the mean square error.

[102]:
```
modelD.compile(loss='mse', optimizer='adam', metrics=[tf.keras.metrics.
↪MeanSquaredError()])
```

The model is fit for only 3 epochs because it quickly overfits the problem. A large batch size of 16 is used. The 33% of train set is used as validation set.

[103]:
```
historyD=modelD.fit([X2_train,X1_train], y_train_scaled, validation_split=0.33,␣
↪epochs=3, batch_size=16,callbacks=[ourCallbackD])
```

```
Epoch 1/3
74/74 [==============================] - 40s 511ms/step - loss: 0.2520 -
mean_squared_error: 0.2520 - val_loss: 0.2071 - val_mean_squared_error: 0.2071
Epoch 2/3
74/74 [==============================] - 25s 341ms/step - loss: 0.1781 -
mean_squared_error: 0.1781 - val_loss: 0.1806 - val_mean_squared_error: 0.1806
Epoch 3/3
```

```
74/74 [==============================] - 25s 342ms/step - loss: 0.1497 -
mean_squared_error: 0.1497 - val_loss: 0.1811 - val_mean_squared_error: 0.1811
```

[104]:
```
#prediction on the test set
predictionsD = modelD.predict([X2_test,X1_test])
```

[105]:
```
#mean square error
mean_squared_error(y_test_scaled,predictionsD)
```

[105]:
```
0.17847966941101395
```

[106]:
```
#R^2 score
r2_score(y_test_scaled,predictionsD)
```

[106]:
```
0.1966494035574905
```

[107]:
```
#explained variance
explained_variance_score(y_test_scaled,predictionsD)
```

[107]:
```
0.21211372026174624
```

[108]:
```
#prediction in days
predictionsD_notscaled=scaler.inverse_transform(predictionsD)
#mse on Number of days
mse_text=mean_squared_error(y_test,predictionsD_notscaled)
```

1.10 Conclusion and possible future work

In this section I will select the best model, for no-text neural networks and discuss possible works related to no-text and text info neural network.

For no text info neural network my best is model C. Comparing the metrics with the others two, model C has the lowest mean square error and the greatest explained variance.In addition model B seems to be overfit, the mean square error computed in the validation set start to increase in the final epochs, before Earlystoppig. Regarding the model D in which I combine numeric variable and text info I obtained a mean square error and a score of explained variance really similar to modelC. One of the problem with recurrent neural networks is that they quick overfit, in fact I have just trained the model for 3 epochs. RNN and LSTM are frequently used with k-fold cross validation. In this case study it's not possible applying k-fold cross validation, but in any further application of RNN with the MIMIC dataset a careful pre-process must be re-think, in order to avoid the impact of the test set on the selection of the features.

The MSE for no text NN in number of days is:

[109]:
```
mse_notext
```

[109]:
```
23678.151747118325
```

The MSE for text and no text NN in number of days is:

[110]:
```
mse_text
```

[110]: 23698.669874194704