Project: (K-) Nearest Neighbors & SVD

Programming project: probability of death

In this project, you have to predict the probability of death of a patient that is entering an ICU (Intensive Care Unit).

The dataset comes from MIMIC project (https://mimic.physionet.org/). MIMIC-III (Medical Information Mart for Intensive Care III) is a large, freely-available database comprising deidentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012.

Each row of *mimic_train.csv* correponds to one ICU stay (*hadm_id+icustay_id*) of one patient (*subject_id*). Column HOSPITAL_EXPIRE_FLAG is the indicator of death (=1) as a result of the current hospital stay; this is the outcome to predict in our modelling exercise. The remaining columns correspond to vitals of each patient (when entering the ICU), plus some general characteristics (age, gender, etc.), and their explanation can be found at *mimic_patient_metadata.csv*.

Please don't use any feature that you infer you don't know the first day of a patient in an ICU.

Note that the main cause/disease of patient condition is embedded as a code at *ICD9_diagnosis* column. The meaning of this code can be found at *MIMIC_metadata_diagnose.csv*. **But** this is only the main one; a patient can have co-occurrent diseases (comorbidities). These secondary codes can be found at *extra_data/MIMIC_diagnoses.csv*.

As performance metric, you can use *AUC* for the binary classification case, but feel free to report as well any other metric if you can justify that is particularly suitable for this case.

Main tasks are:

- Using *mimic_train.csv* file build a predictive model for *HOSPITAL_EXPIRE_FLAG* .
- For this analysis there is an extra test dataset, *mimic_test_death.csv*. Apply your final model to this extra dataset and generate predictions following the same format as *mimic_kaggle_death_sample_submission.csv*. Once ready, you can submit to our Kaggle competition and iterate to improve the accuracy.

As a *bonus*, try different algorithms for neighbor search and for distance, and justify final selection. Try also different weights to cope with class imbalance and also to balance neighbor proximity. Try to assess somehow confidence interval of predictions.

You can follow those **steps** in your first implementation:

1. *Explore* and understand the dataset.
2. Manage missing data.
3. Manage categorial features. E.g. create *dummy variables* for relevant categorical features, or build an ad hoc distance function.
4. Build a prediction model. Try to improve it using methods to tackle class imbalance.
5. Assess expected accuracy of previous models using *cross-validation*.
6. Test the performance on the test file and report accuracy, following same preparation steps (missing data, dummies, etc). Remember that you should be able to yield a prediction for all the rows of the test dataset.

Feel free to reduce the training dataset if you experience computational constraints.

In [ ]:

```
# Mount google drive if running from Google Collab
from google.colab import drive
drive.mount('/content/drive')
```

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

In [ ]:

```
## Set current directory if running from Google Collab

import os
#os.chdir('/content/drive/My Drive/ComputationalML/nearest_neighbors/Pyt
os.chdir('/content/drive/My Drive/CM1_CM2_learning/2021_2022_classes/CM1
```

In [ ]:

```
!pip install category_encoders
```

Generally useful packages

In [ ]:

```
# Imports
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pylab as plt
import seaborn as sns
import sklearn
```

Loading the data

In [ ]:

```
# Training dataset
train = pd.read_csv('bgse-svm-death/mimic_train.csv')
train.head()
```

Out[ ]:

	HOSPITAL_EXPIRE_FLAG	subject_id	hadm_id	icustay_id	HeartRate_Min	HeartRate_
0	0	55440	195768	228357	89.0	1
1	0	76908	126136	221004	63.0	1
2	0	95798	136645	296315	81.0	
3	0	40708	102505	245557	76.0	1
4	0	28424	127337	225281	NaN	

5 rows × 41 columns

In [ ]:

```python
# Test dataset (to produce predictions)
test = pd.read_csv('bgse-svm-death/mimic_test_death.csv')
test.sort_values('icustay_id').head()
```

Out[ ]:

	subject_id	hadm_id	icustay_id	HeartRate_Min	HeartRate_Max	HeartRate_Mean
4930	93535	121562	200011	56.0	82.0	71.205128
1052	30375	177945	200044	NaN	NaN	NaN
3412	73241	149216	200049	54.0	76.0	64.833333
1725	99052	129142	200063	85.0	102.0	92.560976
981	51698	190004	200081	82.0	133.0	94.323529

5 rows × 39 columns

In [ ]:

```python
# Obtaining list of features - train
train.columns
```

Out[ ]:

```
Index(['HOSPITAL_EXPIRE_FLAG', 'subject_id', 'hadm_id', 'icustay_id',
       'HeartRate_Min', 'HeartRate_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'DiasBP_Mea
n',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_Min', 'TempC_Max', 'TempC_
Mean',
       'SpO2_Min', 'SpO2_Max', 'SpO2_Mean', 'Glucose_Min', 'Glucose_Max',
       'Glucose_Mean', 'GENDER', 'DOB', 'ADMITTIME', 'Diff', 'ADMISSION_T
YPE',
       'INSURANCE', 'RELIGION', 'MARITAL_STATUS', 'ETHNICITY', 'DIAGNOSIS
',
       'ICD9_diagnosis', 'FIRST_CAREUNIT', 'LOS'],
      dtype='object')
```

In [ ]:

```
# Obtaining list of features - test
test.columns
```

Out[ ]:

```
Index(['subject_id', 'hadm_id', 'icustay_id', 'HeartRate_Min', 'HeartRate
_Max',
       'HeartRate_Mean', 'SysBP_Min', 'SysBP_Max', 'SysBP_Mean', 'DiasBP_
Min',
       'DiasBP_Max', 'DiasBP_Mean', 'MeanBP_Min', 'MeanBP_Max', 'MeanBP_M
ean',
       'RespRate_Min', 'RespRate_Max', 'RespRate_Mean', 'TempC_Min',
       'TempC_Max', 'TempC_Mean', 'SpO2_Min', 'SpO2_Max', 'SpO2_Mean',
       'Glucose_Min', 'Glucose_Max', 'Glucose_Mean', 'GENDER', 'DOB',
       'ADMITTIME', 'Diff', 'ADMISSION_TYPE', 'INSURANCE', 'RELIGION',
       'MARITAL_STATUS', 'ETHNICITY', 'DIAGNOSIS', 'ICD9_diagnosis',
       'FIRST_CAREUNIT'],
      dtype='object')
```

Dropping columns = 'DOD', 'DISCHTIME', 'DEATHTIME', 'LOS'

In [ ]:

```
# Dropping the different columns from the training data
#extra = ['DOD', 'DISCHTIME', 'DEATHTIME', 'LOS']
extra = ['LOS']
train = train.drop(extra, axis=1)
train.shape
```

Out[ ]:

```
(20885, 40)
```

Do we have missing data?

In [ ]:

```
# Checking for Nulls - train
train.isnull().sum()
```

Out[ ]:

```
HOSPITAL_EXPIRE_FLAG       0
subject_id                 0
hadm_id                    0
icustay_id                 0
HeartRate_Min           2187
HeartRate_Max           2187
HeartRate_Mean          2187
SysBP_Min               2208
SysBP_Max               2208
SysBP_Mean              2208
DiasBP_Min              2209
DiasBP_Max              2209
DiasBP_Mean             2209
MeanBP_Min              2186
MeanBP_Max              2186
MeanBP_Mean             2186
RespRate_Min            2189
RespRate_Max            2189
RespRate_Mean           2189
TempC_Min               2497
TempC_Max               2497
TempC_Mean              2497
SpO2_Min                2203
SpO2_Max                2203
SpO2_Mean               2203
Glucose_Min              253
Glucose_Max              253
Glucose_Mean             253
GENDER                     0
DOB                        0
ADMITTIME                  0
Diff                       0
ADMISSION_TYPE             0
INSURANCE                  0
RELIGION                   0
MARITAL_STATUS           722
ETHNICITY                  0
DIAGNOSIS                  0
ICD9_diagnosis             0
FIRST_CAREUNIT             0
dtype: int64
```

In [ ]:

```
# Checking for Nulls - test
test.isnull().sum()
```

```
Out[ ]: subject_id             0
        hadm_id                0
        icustay_id             0
        HeartRate_Min        545
        HeartRate_Max        545
        HeartRate_Mean       545
        SysBP_Min            551
        SysBP_Max            551
        SysBP_Mean           551
        DiasBP_Min           552
        DiasBP_Max           552
        DiasBP_Mean          552
        MeanBP_Min           547
        MeanBP_Max           547
        MeanBP_Mean          547
        RespRate_Min         546
        RespRate_Max         546
        RespRate_Mean        546
        TempC_Min            638
        TempC_Max            638
        TempC_Mean           638
        SpO2_Min             551
        SpO2_Max             551
        SpO2_Mean            551
        Glucose_Min           58
        Glucose_Max           58
        Glucose_Mean          58
        GENDER                 0
        DOB                    0
        ADMITTIME              0
        Diff                   0
        ADMISSION_TYPE         0
        INSURANCE              0
        RELIGION               0
        MARITAL_STATUS       180
        ETHNICITY              0
        DIAGNOSIS              0
        ICD9_diagnosis         0
        FIRST_CAREUNIT         0
        dtype: int64
```

Do we have class imbalance?

```
In [ ]: # Checking for class imbalance
        train['HOSPITAL_EXPIRE_FLAG'].value_counts()
```

```
Out[ ]: 0    18540
        1     2345
        Name: HOSPITAL_EXPIRE_FLAG, dtype: int64
```

Pre-Processing

Age Variable

Using `ADMITTIME` , `DOB` and `Diff` to create an `age` variable

In [ ]:

```
import datetime as dt

for my_df in [train, test]:
# Convert admittime to date, adding "Diff" to make the dates realistic
    my_df['ADMITTIME'] = (pd.to_datetime(my_df['ADMITTIME']) + my_df["Di
# Convert dob to date, adding "Diff" to make the dates realistic
    my_df['DOB'] = (pd.to_datetime(my_df['DOB']) + my_df["Diff"].apply(la
# Convert to age in years
    my_df['age'] = my_df.apply(lambda e: (e['ADMITTIME'] - e['DOB']).days
```

For patients who are older than 89 years old, we impute the values to be 90 years old. https://github.com/MIT-LCP/mimic-code/issues/637

In [ ]:

```
train['age'] = train.age.where(test['age']<89, 90)
test['age'] = test.age.where(test['age']<89, 90)
```

In [ ]:

```
train = train.drop(['DOB', 'Diff'], axis = 1)
test = test.drop(['DOB', 'Diff'], axis = 1)
```

Combining ethnicities and religion

These are high cardinality categories with very few observations in some. We can logically combine them to reduce the number of dummy variables we have to produce

In [ ]:

```
train['ETHNICITY'].value_counts()
```

Out[ ]:

```
WHITE                                                      15112
BLACK/AFRICAN AMERICAN                                      1977
UNABLE TO OBTAIN                                             577
UNKNOWN/NOT SPECIFIED                                        568
HISPANIC OR LATINO                                           562
OTHER                                                        489
ASIAN                                                        265
PATIENT DECLINED TO ANSWER                                   175
HISPANIC/LATINO - PUERTO RICAN                               155
ASIAN - CHINESE                                              146
BLACK/CAPE VERDEAN                                           126
WHITE - RUSSIAN                                              117
BLACK/HAITIAN                                                 72
HISPANIC/LATINO - DOMINICAN                                   59
ASIAN - ASIAN INDIAN                                          58
WHITE - OTHER EUROPEAN                                        50
MULTI RACE ETHNICITY                                          50
PORTUGUESE                                                    40
WHITE - BRAZILIAN                                             33
ASIAN - VIETNAMESE                                            28
BLACK/AFRICAN                                                 26
MIDDLE EASTERN                                                24
HISPANIC/LATINO - GUATEMALAN                                  24
WHITE - EASTERN EUROPEAN                                      18
HISPANIC/LATINO - CUBAN                                       17
ASIAN - FILIPINO                                              16
ASIAN - CAMBODIAN                                             14
AMERICAN INDIAN/ALASKA NATIVE                                 13
HISPANIC/LATINO - SALVADORAN                                  13
HISPANIC/LATINO - MEXICAN                                      8
HISPANIC/LATINO - CENTRAL AMERICAN (OTHER)                     7
SOUTH AMERICAN                                                 7
CARIBBEAN ISLAND                                               6
ASIAN - KOREAN                                                 6
NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER                      6
ASIAN - JAPANESE                                               6
HISPANIC/LATINO - COLOMBIAN                                    5
ASIAN - OTHER                                                  3
ASIAN - THAI                                                   3
HISPANIC/LATINO - HONDURAN                                     2
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE       2
Name: ETHNICITY, dtype: int64
```

In [ ]:

```
train['RELIGION'].value_counts()
```

Out[ ]:
```
CATHOLIC                     7655
NOT SPECIFIED                5398
PROTESTANT QUAKER            2753
JEWISH                       1840
UNOBTAINABLE                 1515
OTHER                         702
EPISCOPALIAN                  288
GREEK ORTHODOX                178
CHRISTIAN SCIENTIST           164
BUDDHIST                      109
MUSLIM                         74
UNITARIAN-UNIVERSALIST         54
JEHOVAH'S WITNESS              45
ROMANIAN EAST. ORTH            41
HINDU                          38
7TH DAY ADVENTIST              30
HEBREW                          1
Name: RELIGION, dtype: int64
```

By hand

In [ ]:
```
train['ETHNICITY'] = train['ETHNICITY'].replace(['ASIAN', 'ASIAN - CHINES
                                                 'ASIAN - JAPANESE',
                                                 ], 'ASIAN')

train['ETHNICITY'] = train['ETHNICITY'].replace(['HISPANIC OR LATINO', 'H
                                                 'HISPANIC/LATINO - C
                                                 'HISPANIC/LATINO - M
                                                 'HISPANIC/LATINO - H
                                                 ], 'HISPANIC OR LATI

train['ETHNICITY'] = train['ETHNICITY'].replace(['WHITE', 'WHITE - RUSSIA
                                                 'WHITE - BRAZILIAN'
                                                 ], 'WHITE')

train['ETHNICITY'] = train['ETHNICITY'].replace(['BLACK/AFRICAN', 'BLACK/
                                                 ], 'BLACK')

train['ETHNICITY'] = train['ETHNICITY'].replace(['UNABLE TO OBTAIN', 'UNK
                                                 ], 'UNKNOWN')

train['ETHNICITY'] = train['ETHNICITY'].replace(['AMERICAN INDIAN/ALASKA
                                                 'CARIBBEAN ISLAND',
                                                 'MULTI RACE ethnicit
                                                 ], 'OTHER')
```

In [ ]:

```
test['ETHNICITY'] = test['ETHNICITY'].replace(['ASIAN', 'ASIAN - CHINESE
                                               'ASIAN - JAPANESE',
                                               ], 'ASIAN')

test['ETHNICITY'] = test['ETHNICITY'].replace(['HISPANIC OR LATINO', 'HIS
                                               'HISPANIC/LATINO - C
                                               'HISPANIC/LATINO - M
                                               'HISPANIC/LATINO - F
                                               ], 'HISPANIC OR LATI

test['ETHNICITY'] = test['ETHNICITY'].replace(['WHITE', 'WHITE - RUSSIAN
                                               'WHITE - BRAZILIAN'
                                               ], 'WHITE')

test['ETHNICITY'] = test['ETHNICITY'].replace(['BLACK/AFRICAN', 'BLACK/AF
                                               ], 'BLACK')

test['ETHNICITY'] = test['ETHNICITY'].replace(['UNABLE TO OBTAIN', 'UNKNO
                                               ], 'UNKNOWN')

test['ETHNICITY'] = test['ETHNICITY'].replace(['AMERICAN INDIAN/ALASKA NA
                                               'CARIBBEAN ISLAND',
                                               'MULTI RACE ethnicit
                                               ], 'OTHER')
```

In [ ]:

```
religion_other = ['HEBREW', 'UNITARIAN-UNIVERSALIST', 'HINDU', 'GREEK ORT
train['RELIGION'] = train['RELIGION'].replace(religion_other, 'OTHER')
test['RELIGION'] = test['RELIGION'].replace(religion_other, 'OTHER')
```

Or using some string operations

e.g.

```
# Ethnicities list
ethnicities = ["WHITE", "ASIAN", "BLACK", "HISPANIC"]

# Check if the category value contains the word "WHITE",
"ASIAN", "BLACK" or "HISPANIC"
for ethnicity in ethnicities:
    df_train[ethnicity] =
df_train.ETHNICITY.str.contains(ethnicity, regex=False)*1
    df_test[ethnicity] =
df_test.ETHNICITY.str.contains(ethnicity, regex=False)*1
```

Repeat Visits to the ICU

Some patients visited the ICU more than once, we can add the numbe rof previous visits they have had as a variable

In [ ]:

```
train["visits_ICU"] = train.sort_values(['subject_id', 'ADMITTIME']).grou
test["visits_ICU"] = test.sort_values(['subject_id', 'ADMITTIME']).groupb
```

In [ ]:

```
# Print some part of the output
train[["subject_id", "ADMITTIME", "visits_ICU"]].sort_values(["subject_ic
```

Out[ ]:

	subject_id	ADMITTIME	visits_ICU
18310	23	2011-05-02	1
17908	34	2012-07-03	1
9591	36	2011-03-01	1
727	85	2008-09-27	1
16007	109	2008-03-03	1
13738	109	2008-04-30	2
4002	109	2008-08-12	3
12883	109	2008-08-18	4
17983	109	2008-08-25	5
8222	109	2008-09-20	6
1283	109	2008-10-25	7
6733	109	2008-10-31	8
13431	109	2008-11-14	9

In [ ]:

```
train = train.drop(['ADMITTIME'], axis = 1)
test = test.drop(['ADMITTIME'], axis = 1)
```

Diagnoses

- We remove the text fields
- We will target encode (with smoothing) in the pipeline
- Could do something more complicated aggregating the severity of each diagnosis with comorbidites e.g. max, mean, median ect. but for simplicity I stick to this

In [ ]:

```
train = train.drop(['DIAGNOSIS'], axis = 1)
test = test.drop(['DIAGNOSIS'], axis = 1)
```

Number of Comorbidities

A simple way to include the comorbidities is to simply count how many coomorbidities each patient had. One could do more complicated htings also (see above)

In [ ]:

```
# Reading comorbidities dataset
comorbidities = pd.read_csv("bgse-svm-death/extra_data/MIMIC_diagnoses.cs
comorbidities.head()
```

Out[ ]:

	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE
0	256	108811	1.0	53240
1	256	108811	2.0	41071
2	256	108811	3.0	53560
3	256	108811	4.0	40390
4	256	108811	5.0	5859

In [ ]:

```
# Computing the number of comorbidities in each of the ICU stays
number_comorbidities_patient = comorbidities.groupby(["SUBJECT_ID", "HADM
number_comorbidities_patient = number_comorbidities_patient.rename({"SEQ_
number_comorbidities_patient
```

Out[ ]:

	SUBJECT_ID	HADM_ID	number_comorbidities
0	2	163353	3.0
1	3	145834	9.0
2	4	185777	9.0
3	5	178980	3.0
4	6	107064	8.0
...	...	...	...
58971	99985	176670	13.0
58972	99991	151118	17.0
58973	99992	197084	12.0
58974	99995	137810	17.0
58975	99999	113369	5.0

58976 rows × 3 columns

In [ ]:

```
# Join with training set
train = pd.merge(train, number_comorbidities_patient, left_on=["subject_i

# Join with test set
test = pd.merge(test, number_comorbidities_patient, left_on=["subject_id"
```

Handling Missing Values

In [ ]:

```
# Checking for Nulls - train
train.isnull().sum()
```

```
Out[ ]: HOSPITAL_EXPIRE_FLAG        0
        subject_id                  0
        hadm_id                     0
        icustay_id                  0
        HeartRate_Min            2187
        HeartRate_Max            2187
        HeartRate_Mean           2187
        SysBP_Min                2208
        SysBP_Max                2208
        SysBP_Mean               2208
        DiasBP_Min               2209
        DiasBP_Max               2209
        DiasBP_Mean              2209
        MeanBP_Min               2186
        MeanBP_Max               2186
        MeanBP_Mean              2186
        RespRate_Min             2189
        RespRate_Max             2189
        RespRate_Mean            2189
        TempC_Min                2497
        TempC_Max                2497
        TempC_Mean               2497
        SpO2_Min                 2203
        SpO2_Max                 2203
        SpO2_Mean                2203
        Glucose_Min               253
        Glucose_Max               253
        Glucose_Mean              253
        GENDER                      0
        ADMISSION_TYPE              0
        INSURANCE                   0
        RELIGION                    0
        MARITAL_STATUS            722
        ETHNICITY                   0
        ICD9_diagnosis              0
        FIRST_CAREUNIT              0
        age                         0
        visits_ICU                  0
        number_comorbidities        0
        dtype: int64
```

Marital Status

We can logically combine `nan` values for marital status with the `UNKNOWN (DEFAULT)` class. If there were not this class already you could add a 'missing' class

```
In [ ]: train["MARITAL_STATUS"].unique()
```

```
Out[ ]: array(['SINGLE', 'MARRIED', 'SEPARATED', 'WIDOWED', 'DIVORCED', nan,
               'UNKNOWN (DEFAULT)', 'LIFE PARTNER'], dtype=object)
```

```
In [ ]: #first we deal with marital status missing
        # there is a category of unknown, so i'll fill the missing values with tl
        train['MARITAL_STATUS'] = train['MARITAL_STATUS'].fillna('UNKNOWN (DEFAUI
        test['MARITAL_STATUS'] = test['MARITAL_STATUS'].fillna('UNKNOWN (DEFAULT)
```

Forward/Backward filling from previous visits

We can use repeat visits to forward and backward fill missing data

In [ ]:

```
train = train.groupby(['subject_id'], as_index = False).apply(lambda grou
test = test.groupby(['subject_id'], as_index = False).apply(lambda group:
```

In [ ]:

```
# Checking for Nulls - train
train.isnull().sum()
```

Out[ ]:

```
HOSPITAL_EXPIRE_FLAG        0
subject_id                  0
hadm_id                     0
icustay_id                  0
HeartRate_Min            1937
HeartRate_Max            1937
HeartRate_Mean           1937
SysBP_Min                1957
SysBP_Max                1957
SysBP_Mean               1957
DiasBP_Min               1958
DiasBP_Max               1958
DiasBP_Mean              1958
MeanBP_Min               1937
MeanBP_Max               1937
MeanBP_Mean              1937
RespRate_Min             1938
RespRate_Max             1938
RespRate_Mean            1938
TempC_Min                2207
TempC_Max                2207
TempC_Mean               2207
SpO2_Min                 1952
SpO2_Max                 1952
SpO2_Mean                1952
Glucose_Min               214
Glucose_Max               214
Glucose_Mean              214
GENDER                      0
ADMISSION_TYPE              0
INSURANCE                   0
RELIGION                    0
MARITAL_STATUS              0
ETHNICITY                   0
ICD9_diagnosis              0
FIRST_CAREUNIT              0
age                         0
visits_ICU                  0
number_comorbidities        0
dtype: int64
```

In [ ]:

```
train = train.groupby(['subject_id'], as_index = False).apply(lambda grou
test = test.groupby(['subject_id'], as_index = False).apply(lambda group:
```

In [ ]:
```
# Checking for Nulls - train
train.isnull().sum()
```

Out[ ]:
```
HOSPITAL_EXPIRE_FLAG        0
subject_id                  0
hadm_id                     0
icustay_id                  0
HeartRate_Min            1787
HeartRate_Max            1787
HeartRate_Mean           1787
SysBP_Min                1804
SysBP_Max                1804
SysBP_Mean               1804
DiasBP_Min               1805
DiasBP_Max               1805
DiasBP_Mean              1805
MeanBP_Min               1788
MeanBP_Max               1788
MeanBP_Mean              1788
RespRate_Min             1787
RespRate_Max             1787
RespRate_Mean            1787
TempC_Min                2028
TempC_Max                2028
TempC_Mean               2028
SpO2_Min                 1800
SpO2_Max                 1800
SpO2_Mean                1800
Glucose_Min               191
Glucose_Max               191
Glucose_Mean              191
GENDER                      0
ADMISSION_TYPE              0
INSURANCE                   0
RELIGION                    0
MARITAL_STATUS              0
ETHNICITY                   0
ICD9_diagnosis              0
FIRST_CAREUNIT              0
age                         0
visits_ICU                  0
number_comorbidities        0
dtype: int64
```

kNN imputation

We will imputes the reamining continious values of misasing data using kNN as part of the pipeline.

You could bb smarted than this, e.g. presumably age will have a greater impact on vitals that religion will

In [ ]:
```
from sklearn.impute import KNNImputer
from sklearn.impute import SimpleImputer

#cont_imputer = KNNImputer()
cont_imputer = SimpleImputer(strategy="mean")
```

Category Encoding

We will use the pipeline to encode our unordered categorical variables

OneHotEncoding/Dummy variables

`OneHotEncoder` is a slightly nicer alternative to `pd.get_dummies`

`sparse=False` prevents `OneHotEncoder` from outputting a sparse matrix and allowing comptability later down the pipeline.

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder(handle_unknown="ignore", sparse=False)
```

Target Encoding (Smoothed)

We encode each disease by it's deadliness. Smothing can help deal with cases where very observations had a certain disease.

$$X_{ICD9-TE,j} = \lambda \frac{\#\text{ died with disease } j}{\#\text{ with disease } j} + (1-\lambda)\frac{\#\text{ died}}{\#\text{ patients}}$$

In [ ]:
```
import category_encoders as ce

icd9_encoder = ce.TargetEncoder(smoothing = 1.0)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in th
e public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Scaling

I am going to use a RobustScaler for the continious variabes and StandardScalar for the discrete ones

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler

cont_scaler = RobustScaler()
cat_scaler = StandardScaler()
```

In [ ]:

```
X_train = train.drop(["HOSPITAL_EXPIRE_FLAG",   "subject_id",   "hadm_id'
y_train = train["HOSPITAL_EXPIRE_FLAG"]
```

In [ ]:

```
X_train
```

Out[ ]:

	HeartRate_Min	HeartRate_Max	HeartRate_Mean	SysBP_Min	SysBP_Max	SysBP
0	89.0	145.0	121.043478	74.0	127.0	106.5
1	63.0	110.0	79.117647	89.0	121.0	106.7
2	81.0	98.0	91.689655	88.0	138.0	112.
3	76.0	128.0	98.857143	84.0	135.0	106.9
4	58.0	64.0	60.324324	78.0	118.0	99.
...	...	...	...	...	...	
20880	65.0	92.0	78.500000	60.0	160.0	110.9
20881	74.0	112.0	89.156250	100.0	150.0	123.
20882	58.0	97.0	76.933333	94.0	131.0	112.0
20883	59.0	102.0	81.844444	96.0	150.0	123.8
20884	59.0	97.0	77.526316	82.0	139.0	106.

20885 rows × 35 columns

In [ ]:

```
y_train
```

Out[ ]:

```
0        0
1        0
2        0
3        0
4        0
        ..
20880    0
20881    0
20882    0
20883    0
20884    0
Name: HOSPITAL_EXPIRE_FLAG, Length: 20885, dtype: int64
```

Preprocessing Pipeline

We will use the `ColumnTransformer` to allwo for different data preprocessing for differen types of columns

- Numerical values - `RobustScaler()` and `KNNImputer()`
- Categoricals (not ICD9) - `OneHotEncoder()` and `StandardScaler()`
- ICD9 - `TargetEncoder()` and `StandardScaler()`

In [ ]:

```
# Update list of numerical and categorical features
num_feat = X_train.select_dtypes(exclude=['object', 'category']).columns
print(num_feat)

cat_feat = X_train.select_dtypes(include=['object', 'category']).columns

# make own category for preprocessing 'ICD9_diagnosis'
icd9_feat = ['ICD9_diagnosis']
cat_feat = cat_feat.drop(['ICD9_diagnosis'])
print(icd9_feat)
print(cat_feat)
```

```
Index(['HeartRate_Min', 'HeartRate_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'DiasBP_Mea
n',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_Min', 'TempC_Max', 'TempC_
Mean',
       'SpO2_Min', 'SpO2_Max', 'SpO2_Mean', 'Glucose_Min', 'Glucose_Max',
       'Glucose_Mean', 'age', 'visits_ICU', 'number_comorbidities'],
      dtype='object')
['ICD9_diagnosis']
Index(['GENDER', 'ADMISSION_TYPE', 'INSURANCE', 'RELIGION', 'MARITAL_STAT
US',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')
```

In [ ]:

```
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.compose import ColumnTransformer



# pipeline for numerical data
num_preprocessing = make_pipeline(
    cont_scaler,
    cont_imputer
    )

# pipeline for categorical data
cat_preprocessing = make_pipeline(
    #SimpleImputer(strategy="most_frequent"), # we only have missing data
    cat_encoder,
    cat_scaler)

icd9_preprocessing = make_pipeline(
    icd9_encoder,
    cat_scaler)



# combine preprocessing pipelines using a columnTransformer
preprocessing = ColumnTransformer(
    [("num", num_preprocessing, num_feat),
     ("cat", cat_preprocessing, cat_feat),
     ("icd9", icd9_preprocessing, icd9_feat)
     ]
    #,remainder='passthrough'
    , remainder='drop'
)
```

In [ ]:

```
from sklearn import set_config
set_config(display="diagram")
preprocessing
```

Out[ ]:

```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('robustscaler',
                                                  RobustScaler()),
                                                 ('simpleimputer',
                                                  SimpleImputer())
]),
                                 Index(['HeartRate_Min', 'HeartRat
e_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_M...
                                 Pipeline(steps=[('onehotencoder',
                                                  OneHotEncoder(ha
ndle_unknown='ignore',
                                                                sp
arse=False)),
                                                 ('standardscaler'
,
```

```
                                                    StandardScaler()
)]),
                                  Index(['GENDER', 'ADMISSION_TYPE'
, 'INSURANCE', 'RELIGION', 'MARITAL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')),
                                 ('icd9',
                                  Pipeline(steps=[('targetencoder',
                                                    TargetEncoder())
,
                                                   ('standardscaler'
,
                                                    StandardScaler()
)]),
                                  ['ICD9_diagnosis'])])
```

Please rerun this cell to show the HTML repr or trust the notebook.

ColumnTransformer

```
ColumnTransformer(transformers=[('num',
                                  Pipeline(steps=[('robustscaler',
                                                    RobustScaler()),
                                                   ('simpleimputer',
                                                    SimpleImputer())
]),
                                  Index(['HeartRate_Min', 'HeartRat
e_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_M...
                                  Pipeline(steps=[('onehotencoder',
                                                    OneHotEncoder(ha
ndle_unknown='ignore',
                                                                  sp
arse=False)),
                                                   ('standardscaler'
,
                                                    StandardScaler()
)]),
                                  Index(['GENDER', 'ADMISSION_TYPE'
, 'INSURANCE', 'RELIGION', 'MARITAL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')),
                                 ('icd9',
                                  Pipeline(steps=[('targetencoder',
                                                    TargetEncoder())
,
                                                   ('standardscaler'
,
                                                    StandardScaler()
)]),
                                  ['ICD9_diagnosis'])])
```

num

```
Index(['HeartRate_Min', 'HeartRate_Max', 'HeartRate_Mean', 'SysBP_
```

```
Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_Min', 'TempC_Max',
'TempC_Mean',
       'SpO2_Min', 'SpO2_Max', 'SpO2_Mean', 'Glucose_Min', 'Glucos
e_Max',
       'Glucose_Mean', 'age', 'visits_ICU', 'number_comorbidities'
],
      dtype='object')
```

RobustScaler

```
RobustScaler()
```

SimpleImputer

```
SimpleImputer()
```

cat

```
Index(['GENDER', 'ADMISSION_TYPE', 'INSURANCE', 'RELIGION', 'MARIT
AL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')
```

OneHotEncoder

```
OneHotEncoder(handle_unknown='ignore', sparse=False)
```

StandardScaler

```
StandardScaler()
```

icd9

```
['ICD9_diagnosis']
```

TargetEncoder

```
TargetEncoder()
```

StandardScaler

```
StandardScaler()
```

In [ ]:

```
#X_train_pp = preprocessing.fit_transform(X_train, y_train)
```

In [ ]:

```
preprocessing.fit(X_train, y_train)
X_train_pp = preprocessing.transform(X_train)
```

In [ ]:

```
X_train_pp.shape
```

Out[ ]:

```
(20885, 63)
```

Class Imbalance

- The `SVC` offers the `class_weight = 'balanced'`, this conducts `RandomOverSampling` of the minority class till the classes are balanced.
- `KNeighborsClassifier` has no such option
- and in any case randomly repeating minority observations is not necessarily the best you can do

SMOTEing

SMOTE = Synthetic Minority Oversampling Technique. The idea is to creat synthetic observations that are lie inbetween two close member so fthe minority class

- Sample minority observation at random
- Sample one of k nearets neighbours (default $k = 5$)
- Draw a line between the point and the chosen neighbour
- Generate a new observation on this line

Tomek links

Tomek links is a method for undersampling. This removes majority class observations that are close to minoity class observations.

x_i and x_j are Tomek links if

- x_i is x_j's nearets neighbour
- x_j is x_i's nearest neighbour
- $y_i \neq y_j$

https://imbalanced-learn.org/stable/references/generated/imblearn.combine.SMOTETomek.html

In [ ]:

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
```

kNN

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from imblearn.pipeline import Pipeline as imbPipe

kNN_pipe = imbPipe([
          ('preprocess', preprocessing),
        #('oversampling', SMOTE()),
        #('undersampling', RandomUnderSampler()),
        #('resampling', SMOTETomek(tomek=TomekLinks(sampling_strategy='ma
        #('features', fs.RFECV(estimator = DecisionTreeClassifier(class_w
        #                              step = 10, cv = 5, scoring = 'roc_a
        ('kNN', KNeighborsClassifier(algorithm = 'auto'))
])
```

In [ ]:

```
from sklearn import set_config
set_config(display="diagram")
kNN_pipe
```

Out[ ]:

```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[
('robustscaler',

RobustScaler()),

('simpleimputer',

SimpleImputer())]),
                                                  Index(['HeartRat
e_Min', 'HeartRate_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate...

OneHotEncoder(handle_unknown='ignore',

sparse=False)),

('standardscaler',

StandardScaler())]),
                                                  Index(['GENDER',
'ADMISSION_TYPE', 'INSURANCE', 'RELIGION', 'MARITAL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')),
                                                 ('icd9',
                                                  Pipeline(steps=[
('targetencoder',

TargetEncoder()),

('standardscaler',
```

```
StandardScaler())]),
                                                  ['ICD9_diagnosis
'])])),
                ('kNN', KNeighborsClassifier())])
```

Please rerun this cell to show the HTML repr or trust the notebook.

Pipeline

```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[
('robustscaler',

RobustScaler()),

('simpleimputer',

SimpleImputer())]),
                                                  Index(['HeartRat
e_Min', 'HeartRate_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate...

OneHotEncoder(handle_unknown='ignore',

sparse=False)),

('standardscaler',

StandardScaler())]),
                                                  Index(['GENDER',
'ADMISSION_TYPE', 'INSURANCE', 'RELIGION', 'MARITAL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')),
                                                 ('icd9',
                                                  Pipeline(steps=[
('targetencoder',

TargetEncoder()),

('standardscaler',

StandardScaler())]),
                                                  ['ICD9_diagnosis
'])])),
                ('kNN', KNeighborsClassifier())])
```

preprocess: ColumnTransformer

```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('robustscaler',
                                                  RobustScaler()),
                                                 ('simpleimputer',
                                                  SimpleImputer())
]),
```

```
                                 Index(['HeartRate_Min', 'HeartRat
e_Max', 'HeartRate_Mean', 'SysBP_Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_M...
                                 Pipeline(steps=[('onehotencoder',
                                                  OneHotEncoder(ha
ndle_unknown='ignore',
                                                                sp
arse=False)),
                                                 ('standardscaler'
,
                                                  StandardScaler()
)]),
                                 Index(['GENDER', 'ADMISSION_TYPE'
, 'INSURANCE', 'RELIGION', 'MARITAL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')),
                                ('icd9',
                                 Pipeline(steps=[('targetencoder',
                                                  TargetEncoder())
,
                                                 ('standardscaler'
,
                                                  StandardScaler()
)]),
                                 ['ICD9_diagnosis'])])
```
num
```
Index(['HeartRate_Min', 'HeartRate_Max', 'HeartRate_Mean', 'SysBP_
Min',
       'SysBP_Max', 'SysBP_Mean', 'DiasBP_Min', 'DiasBP_Max', 'Dia
sBP_Mean',
       'MeanBP_Min', 'MeanBP_Max', 'MeanBP_Mean', 'RespRate_Min',
       'RespRate_Max', 'RespRate_Mean', 'TempC_Min', 'TempC_Max',
'TempC_Mean',
       'SpO2_Min', 'SpO2_Max', 'SpO2_Mean', 'Glucose_Min', 'Glucos
e_Max',
       'Glucose_Mean', 'age', 'visits_ICU', 'number_comorbidities'
],
      dtype='object')
```
RobustScaler
```
RobustScaler()
```
SimpleImputer
```
SimpleImputer()
```
cat
```
Index(['GENDER', 'ADMISSION_TYPE', 'INSURANCE', 'RELIGION', 'MARIT
AL_STATUS',
       'ETHNICITY', 'FIRST_CAREUNIT'],
      dtype='object')
```
OneHotEncoder
```
OneHotEncoder(handle_unknown='ignore', sparse=False)
```
StandardScaler

StandardScaler()
icd9
['ICD9_diagnosis']
TargetEncoder
TargetEncoder()
StandardScaler
StandardScaler()
KNeighborsClassifier
KNeighborsClassifier()

GridSearch

The minimum I wanted to see you grid searh over were the number of neighbours, the 'weights' and the parameter sof the distance.

Some students also considered specifically defining the `metric` .

Once idea I particularly liked was deliberatley scaling different colluns differently to give them higher or lower importance

In [ ]:

```
kNN_params = {#'preprocess__num__cont_imputer__n_neighbors':[10, 50, 100
              #'preprocess__num__cont_imputer__strategy':["mean", "media
              #'preprocess__icd9__icd9_encoder__smoothing':[0.5, 1, 2]
              #'oversampling__sampling_strategy':[0.1,0.3,0.5],
              #'undersampling__sampling_strategy':[0.5, 0.6, 0.7],
              #'kNN__n_neighbors':[1, 2, 5, 10, 20, 50],
              'kNN__n_neighbors':[300],
              #'kNN__weights':['uniform', 'distance'],
              'kNN__weights':['distance'],
              #'kNN__p':[1, 2, 3]
              'kNN__p':[3]
              }
```

Here I consider `HalvingGridSearch()` to speed up my grid search.

In [ ]:

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

In [ ]:

```
grid_kNN = HalvingGridSearchCV(kNN_pipe, kNN_params, scoring='roc_auc', c

grid_kNN.fit(X_train, y_train)

print("Best parameter (CV score=%0.3f):" % grid_kNN.best_score_)
print(grid_kNN.best_params_)
```

```
Best parameter (CV score=0.770):
{'kNN__n_neighbors': 300, 'kNN__p': 3, 'kNN__weights': 'distance'}
```

Test set predictions

In [ ]:

```
X_test = test.drop(["subject_id",       "hadm_id",      "icustay_id"], ax
```

In [ ]:

```
y_pred_kNN = grid_kNN.predict_proba(X_test)
```

Reweighting to adjust for class-imbalance

In [ ]:

```
def reweight(pi,q1=0.5,r1=0.5):
    r0 = 1-r1
    q0 = 1-q1
    tot = pi*(q1/r1)+(1-pi)*(q0/r0)
    w = pi*(q1/r1)
    w /= tot
    return w
## assign q the proper reweight
q1 = y_test.sum()/len(y_test)
r1 = 0.5 #?? this will depend on what reweighting you did
## reweight probabilites
y_pred_kNN_reweighted = pd.Series(y_pred_kNN[:,1]).apply(reweight,args=(c
```

or

Calibrating Probabilities

`LogisticRegressoon()` built a model for $P(Y|X, \beta)$ and then trained this model to

$$\max \sum_{i=1}^{n} \log P(Y = y_i | X_i, \beta)$$

this can be shown to trained the model to produce *calibrared* probabilities.

For a binary event A, the probability p_A is well calibrated if $p_A \times 100\%$ **of the time that you quote p_A, A happens**

`CalibratedClassifierCV` uses cross-validation both to fit the model and also to assess its calibration and recalibrate it.

https://scikit-learn.org/stable/modules/calibration.html#calibration

In [ ]:

```
# Calibrated probabilities of the best model
kNN_calibrator = CalibratedClassifierCV(grid_kNN, cv = 5, method = 'isotc
kNN_calibrator.fit(X_train, y_train)
```

In [ ]:

```
y_pred_kNN_recalibrated = kNN_calibrator.fit(X_test)
```

Comparing to the true y's

While you could submit to kaggle I can use the real y's

In [ ]:

```
# Training dataset
y_test_true = pd.read_csv('mimic_test_death_true.csv')
```

In [ ]:

```
y_test_true
```

In [ ]:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test_true["HOSPITAL_EXPIRE_FLAG"], y_pred_kNN[:,1])
#roc_auc_score(y_test_true, y_pred_kNN_reweighted)
#roc_auc_score(y_test_true, y_pred_kNN_recalibrated[:,1])
```

SVM

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from imblearn.pipeline import Pipeline as imbPipe

SVM_pipe = imbPipe([
          ('preprocess', preprocessing),
        #('oversampling', SMOTE()),
        #('undersampling', RandomUnderSampler()),
        #('resampling', SMOTETomek(tomek=TomekLinks(sampling_strategy='ma
        #('features', fs.RFECV(estimator = DecisionTreeClassifier(class_w
        #                              step = 10, cv = 5, scoring = 'roc_a
        ('SVC', SVC()
)])
```

In [ ]:

```
from sklearn import set_config
set_config(display="diagram")
SVM_pipe
```

Gridsearch

In [ ]:

```
SVM_params = {#'preprocess__num__cont_imputer__n_neighbors':[10, 50, 100
          #'preprocess__icd9__icd9_encoder__smoothing':[0.5, 1, 2]
          #'oversampling__sampling_strategy':[0.1,0.3,0.5],
          #'undersampling__sampling_strategy':[0.5, 0.6, 0.7],
          #'SVC__C':[0.1, 1, 10],
          'SVC__C':[1],
          #'SVC__kernel':['linear', 'rbf', 'poly],
          'SVC__kernel':['rbf'],
          #'SVC__gamma':[0.25, 0.5, 0.75]
          'SVC__gamma':[0.5]
          }
```

In [ ]:

```
grid_SVM = HalvingGridSearchCV(SVM_pipe, SVM_params, scoring='roc_auc', c

grid_SVM.fit(X_train, y_train)

print("--- %s seconds ---" % (time.time() - start_time))
print("Best parameter (CV score=%0.3f):" % grid_SVM.best_score_)
print(grid_SVM.best_params_)
```