1 Lab 1: Risk stratification using MIMIC

MIMIC-III is the largest publically available clinical dataset. Here we seek predict in-hospital mortality using data about the patient’s admission to the ICU.

In [1]: import datetime
   import numpy as np
   import pandas as pd
   import psycopg2

   Let’s set up our connection to our MIMIC postgres server. For more information on installing MIMIC, see the official documentation.

   In [2]: sqluser = 'dsontag' # change this to whatever your username is
doename = 'mimic'
schema_name = 'mimiciii'

   # Connect to local postgres version of mimic
   con = psycopg2.connect(dbname=dbname, user=sqluser)
cur = con.cursor()
cur.execute('SET search_path to ' + schema_name)

   In [4]: query = 
   """
   select *
   from admissions
   """

   admissions_df = pd.read_sql_query(query, con)
admissions_df.head()

Out[4]:   row_id  subject_id  hadm_id   admittime   dischtime
       0      21        22  165315  2196-04-09 12:26:00  2196-04-10 15:54:00
       1      22        23  152223  2153-09-03 07:15:00  2153-09-08 19:10:00
       2      23        23  124321  2157-10-18 19:34:00  2157-10-25 14:00:00
       3      24        24  161859  2139-06-06 16:14:00  2139-06-09 12:48:00
Much of the information we are interested in is in the patients table, such as the date of birth. We need to load it. Call the loaded patients table ‘patients_df’

```python
In [5]: query = 
   
   ""
   select * from patients
   ""

patients_df = pd.read_sql_query(query, con)
patients_df.head()
```

```
Out[5]:  row_id  subject_id  gender  dob  dod  dod_hosp  dod_ssn  
0      4       25     25  129635  2160-11-02 02:06:00  2160-11-05 14:55:00
1      1      25     25  129635  2160-11-02 02:06:00  2160-11-05 14:55:00
2      2      25     25  129635  2160-11-02 02:06:00  2160-11-05 14:55:00
3      3      25     25  129635  2160-11-02 02:06:00  2160-11-05 14:55:00
4      4      25     25  129635  2160-11-02 02:06:00  2160-11-05 14:55:00
```
2) Next we need to merge the two tables using the field subject_id, which is shared across both tables.

In [6]: len(patients_df)
Out[6]: 46520

In [7]: len(admissions_df)
Out[7]: 58976

In [8]: combined_df = admissions_df.merge(patients_df, on='subject_id')

In [11]: combined_df.head()

Out[11]:
   row_id_x  subject_id  hadm_id  admittime  dischtime  deathtime admission_type admission_location
0        21       22  165315  2196-04-09 12:26:00  2196-04-10 15:54:00      NaT     EMERGENCY     EMERGENCY ROOM ADMIT
1        22       23  152223  2153-09-03  07:15:00  2153-09-08 19:10:00      NaT     ELECTIVE     PHYS REFERRAL/NORMAL DELI
2        23       23  124321  2157-10-18 19:34:00  2157-10-25 14:00:00      NaT     EMERGENCY     TRANSFER FROM HOSP/EXTRAM
3        24       24  161859  2139-06-06 16:14:00  2139-06-09 12:48:00      NaT     EMERGENCY     TRANSFER FROM HOSP/EXTRAM
4        25       25  129635  2160-11-02 02:06:00  2160-11-05 14:55:00      NaT     EMERGENCY     EMERGENCY ROOM ADMIT

   discharge_location  insurance
0  DISC-TRAN CANCER/CHLDRN H   Private
1          HOME HEALTH CARE  Medicare
2          HOME HEALTH CARE  Medicare
3            HOME     Private
4            HOME     Private
If we want to see the patient’s age at admission, then we need to subtract the admission time from the date of birth. Working with dates and times is tricky in Python, so we write a function to compute the age. Note that subtracting two datetimes will give us the distance in seconds, and we divide appropriately.

```python
In [12]: def get_age(dob, admittime):
    diff = (admittime - dob).total_seconds() / (3600 * 24 * 365.25)
    return diff

combined_df['age'] = combined_df.apply(lambda x: get_age(x['dob'], x['admittime']), axis=1)
```

```
Out[13]:
0       64.926812
1       71.130191
2       75.254799
3       39.016226
4       58.948905
Name: age, dtype: float64
```

3) We now define the features. Let’s start with ‘admission_type’, ‘admission_location’, ‘insurance’, and ‘marital_status’.

Note that because all of our features are categorical, we need to binarize our data using `get_dummies` which transforms a categorical feature of `x = ["a", "b", "a"]` to `x = [[1, 0], [0,1], [1,0]]` where the columns are a, b.

Combine the features into a single data frame called X.
In [80]: X1 = pd.get_dummies(combined_df['admission_type'], prefix='adm')
X2 = pd.get_dummies(combined_df['admission_location'], prefix='loc')
X3 = pd.get_dummies(combined_df['insurance'], prefix='insur')
X4 = pd.get_dummies(combined_df['marital_status'], prefix='marital')
X5 = pd.get_dummies(combined_df['ethnicity'], prefix='eth')
X6 = pd.get_dummies(combined_df['gender'], prefix='gender')
X7 = combined_df['age']

X = pd.concat([X1, X2, X3, X4, X5, X6, X7], axis=1)

In [81]: X.head()

Out[81]:
adm_ELECTIVE  adm_EMERGENCY  adm_NEWBORN  adm_URGENT  \
0          0              1           0           0  
1          1              0           0           0  
2          0              1           0           0  
3          0              1           0           0  
4          0              1           0           0  
loc_** INFO NOT AVAILABLE **  loc_CLINIC REFERRAL/PREMATURE  \
0              0              0              0  
1              0              0              0  
2              0              0              0  
3              0              0              0  
4              0              0              0  
loc_EMERGENCY ROOM ADMIT  loc_HMO REFERRAL/SICK  \
0              1              0              0  
1              0              0              0  
2              0              0              0  
3              0              0              0  
4              1              0              0  
loc_PHYS REFERRAL/NORMAL DELI  loc_TRANSFER FROM HOSP/EXTRAM  ...  \
0              0              0              ...  
1              1              0              ...  
2              0              1              ...  
3              0              1              ...  
4              0              0              ...  
eth_UNABLE TO OBTAIN  eth_UNKNOWN/NOT SPECIFIED  eth_WHITE  \
0              0              0              1  
1              0              0              1  
2              0              0              1  
3              0              0              1  
4              0              0              1
Define the outcome $y$ to be hospital_expire_flag, which is 1 if a patient dies or 0 if not. Note that we don’t differentiate out types of death (in hopsital, in-ICU) and we are only using admissions numbers.

In [39]: y = combined_df['hospital_expire_flag']
y.head()

Out[39]:    0   0
         1   0
         2   0
         3   0
         4   0
Name: hospital_expire_flag, dtype: int64

In [42]: y.describe()

Out[42]:
    count    58976.000000
     mean      0.099261
      std      0.299014
     min       0.000000
   25%        0.000000
   50%        0.000000
   75%        0.000000
     max       1.000000
Name: hospital_expire_flag, dtype: float64
5) Lastly we train a logistic regression using standard ML techniques like splitting into train and test sets. We will be interested in computing the Area under the ROC curve (AUC), so we need to compute the predicted probability and compare to the true label.

In [43]: from sklearn.linear_model import LogisticRegression

In [82]: from sklearn.model_selection import train_test_split
   : Xtrain, Xtest, ytrain, ytest = train_test_split(X,y, train_size=.8)

In [94]: clf = LogisticRegression(random_state=0, penalty='l1', C=.1)
   : clf.fit(Xtrain,ytrain)

Out[94]: LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
   : intercept_scaling=1, max_iter=100, multi_class='warn',
   : n_jobs=None, penalty='l1', random_state=0, solver='warn',
   : tol=0.0001, verbose=0, warm_start=False)

In [95]: from sklearn.metrics import roc_auc_score
   : ypred = clf.predict_proba(Xtest)[:,1]
   : roc_auc_score(ytest, ypred)

Out[95]: 0.7110272535123401

6) How does each feature contribute to a person’s likelihood of dying in the hospital? We can examine the LR coefficients for that.

In [96]: for i,j in sorted(zip(X.columns,clf.coef_[0]), key=lambda x: x[1]):
   :     if not (j == 0):
   :         print i,j

adm_NEWBORN -2.4267307589126865
adm_ELECTIVE -1.1755898822361552
marital_SINGLE -0.7730645498339617
marital_DIVORCED -0.5800296234119904
marital_WIDOWED -0.4966591712420658
marital_MARRIED -0.4884464182403196
insur_Government -0.427268099193038
loc_PHYS REFERRAL/NORMAL DELI -0.42278692478541685
marital_SEPARATED -0.3964586040277649
insur_Private -0.3202025018128261
eth_BLACK/AFRICAN AMERICAN -0.2653369873466112
insur_Medicaid -0.2297739662096113
eth_HISPANIC OR LATINO -0.22672756569519503
loc_CLINIC REFERRAL/PREMATURE -0.20221963089335637
loc_TRANSFER FROM HOSP/EXTRAM -0.10889465544769228
gender_M -0.014077399646318033
age 0.002764580782990953
eth_ASIA 0.030362823104629753
adm_EMERGENCY 0.0760226262798542
insur_Medicare 0.10400368299041315
eth_UNKNOWN/NOT SPECIFIED 0.40800914177933933
eth_UNABLE TO OBTAIN 0.511061585478278