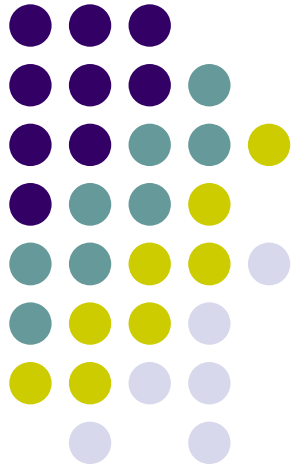
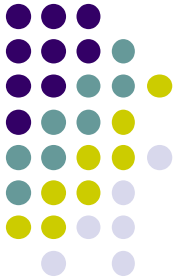
Predicting Hospital Readmission with Discharge Summaries

https://colab.research.google.com/drive/1RRcVFVEq56_C8DzAxCe5gsF3F_Z3bZOn?usp=sharing



This Tutorial

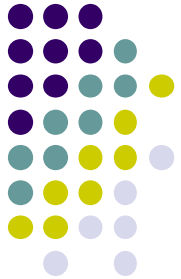


- https://towardsdatascience.com/introduction-to-clinical-natural-language-processing-predicting-hospital-readmission-with-1736d52bc709

- https://github.com/andrewwlong/mimic_bow/blob/master/mimic_bow_full.ipynb

1. Background

Introduction



Clinical notes are important data in EHR, but few can take advantage of them because NLP technologies are required to process these textual data.

These notes represent a vast wealth of knowledge and insight that can be utilized for predictive models using Natural Language Processing (NLP) to improve patient care and hospital workflow.

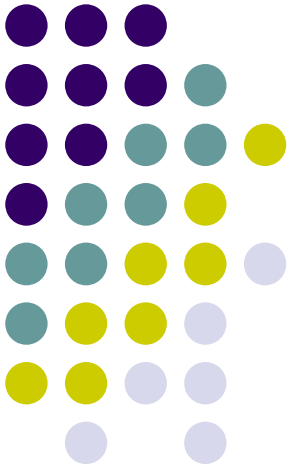
This tutorial will show you how to predict hospital readmission using discharge summaries.

You will learn

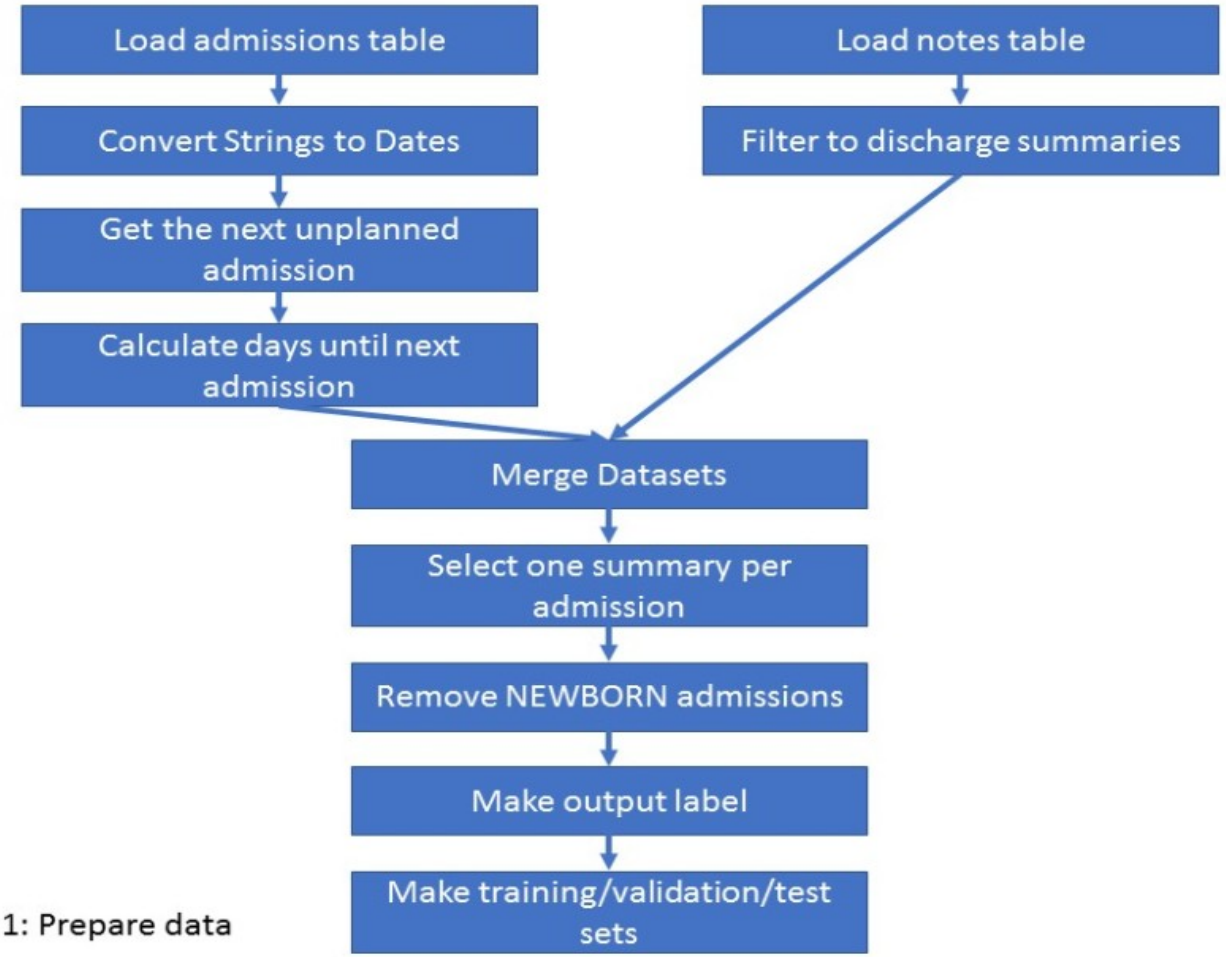
- How to prepare data for a machine learning project
- How to preprocess the unstructured notes using a bag-of-words approach
- How to build a simple predictive model
- How to assess the quality of your model
- How to decide the next step for improving the model

Data and pre-processing

We will utilize the MIMIC-III (Medical Information Mart for Intensive Care III) database.

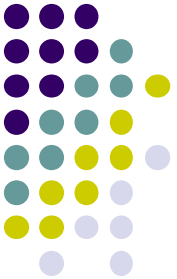


Step 1. Prepare data



Step 1: Prepare data

Import



```
[ ] %matplotlib inline
    # https://towardsdatascience.com/introduction-to-clinical-natural-language-processing-predicting-hospital-readmission-with-1736d52bc709

    # !kill -9 -1
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import nltk
    from nltk import word_tokenize
    import os.path
    import string
```

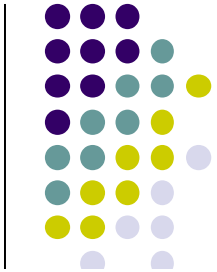
Load Admissions Table

```
[ ]  from google.colab import files
     uploaded = files.upload()
```

- **ADMISSIONS.csv.gz**(application/x-gzip) - 2525254 bytes, last modified: 8/27/2019 - 100% done

```
Saving ADMISSIONS.csv.gz to ADMISSIONS.csv.gz
```

Load admission table using pandas



Convert strings to dates

A note about dates from MIMIC website:

> All dates in the database have been shifted to protect patient confidentiality. Dates will be internally consistent for the same patient, but randomly distributed in the future.

When converting dates, it is safer to use a format. For references on formats see http://strftime.org/. Errors = 'coerce' allows NaT (not a datetime) to happen when the string doesn't match the format

```
df_adm = pd.read_csv('ADMISSIONS.csv')
```

```
# convert to dates
df_adm.ADMITTIME = pd.to_datetime(df_adm.ADMITTIME, format = '%Y-%m-%d %H:%M:%S', errors = 'coerce')
df_adm.DISCHTIME = pd.to_datetime(df_adm.DISCHTIME, format = '%Y-%m-%d %H:%M:%S', errors = 'coerce')
df_adm.DEATHTIME = pd.to_datetime(df_adm.DEATHTIME, format = '%Y-%m-%d %H:%M:%S', errors = 'coerce')
```

```
# check to see if there are any missing dates
print('Number of missing date admissions:', df_adm.ADMITTIME.isnull().sum())
print('Number of missing date discharges:', df_adm.DISCHTIME.isnull().sum())
```

```
Number of missing date admissions: 0
Number of missing date discharges: 0
```

```
df_adm.groupby(['ADMISSION_TYPE']).size()
```

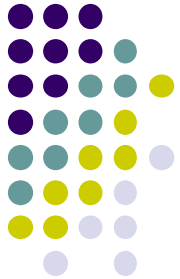
```
ADMISSION_TYPE
ELECTIVE      7706
EMERGENCY    42071
NEWBORN       7863
URGENT        1336
dtype: int64
```

Get the next admission date if it exists

In this project, we need the next admission date if it exists. We can get this with the shift() function, but we need to verify the dates are in order.

```
# sort by subject_ID and admission date
df_adm = df_adm.sort_values(['SUBJECT_ID','ADMITTIME'])
df_adm = df_adm.reset_index(drop = True)
```

	ROW_ID	SUBJECT_ID	HADM_ID	ADMITTIME	DISCHTIME	DEATHTIME	ADMISSION_TYPE	ADMISSION_LOCATION	DISCHARGE_LOCATION
0	1	2	163353	2138-07-17 19:04:00	2138-07-21 15:48:00	NaT	NEWBORN	PHYS REFERRAL/NORMAL DELI	HOME
1	2	3	145834	2101-10-20 19:08:00	2101-10-31 13:58:00	NaT	EMERGENCY	EMERGENCY ROOM ADMIT	SNF
2	3	4	185777	2191-03-16 00:28:00	2191-03-23 18:41:00	NaT	EMERGENCY	EMERGENCY ROOM ADMIT	HOME WITH HOME IV PROVIDR
3	4	5	178980	2103-02-02 04:31:00	2103-02-04 12:15:00	NaT	NEWBORN	PHYS REFERRAL/NORMAL DELI	HOME
4	5	6	107064	2175-05-30 07:15:00	2175-06-15 16:00:00	NaT	ELECTIVE	PHYS REFERRAL/NORMAL DELI	HOME HEALTH CARE



```
# verify that it did what we wanted
df_adm.loc[df_adm.SUBJECT_ID == 124,['SUBJECT_ID','ADMITTIME','ADMISSION_TYPE']]
```

	SUBJECT_ID	ADMITTIME	ADMISSION_TYPE
165	124	2160-06-24 21:25:00	EMERGENCY
166	124	2161-12-17 03:39:00	EMERGENCY
167	124	2165-05-21 21:02:00	ELECTIVE
168	124	2165-12-31 18:55:00	EMERGENCY

```
# add the next admission date and type for each subject using groupby
# you have to use groupby otherwise the dates will be from different subjects
df_adm['NEXT_ADMITTIME'] = df_adm.groupby('SUBJECT_ID').ADMITTIME.shift(-1)
# get the next admission type
df_adm['NEXT_ADMISSION_TYPE'] = df_adm.groupby('SUBJECT_ID').ADMISSION_TYPE.shift(-1)
```

```
# verify that it did what we wanted
df_adm.loc[df_adm.SUBJECT_ID == 124,['SUBJECT_ID','ADMITTIME','ADMISSION_TYPE','NEXT_ADMITTIME','NEXT_ADMISSION_TYPE']]
```

	SUBJECT_ID	ADMITTIME	ADMISSION_TYPE	NEXT_ADMITTIME	NEXT_ADMISSION_TYPE
165	124	2160-06-24 21:25:00	EMERGENCY	2161-12-17 03:39:00	EMERGENCY
166	124	2161-12-17 03:39:00	EMERGENCY	2165-05-21 21:02:00	ELECTIVE
167	124	2165-05-21 21:02:00	ELECTIVE	2165-12-31 18:55:00	EMERGENCY
168	124	2165-12-31 18:55:00	EMERGENCY	NaT	NaN

Shift() function

```
df_bike_rides['previous_ride_end_time'] = df_bike_rides['ride_end_time'].shift(periods=1)
```

View `df_bike_rides`.

```
df_bike_rides
```

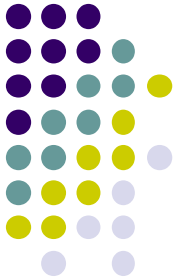
	ride_start_time	ride_end_time	previous_ride_end_time
0	2019-04-21 21:23:29.711347	2019-04-21 21:41:29.711347	NaT
1	2019-04-21 22:43:29.711347	2019-04-21 22:51:29.711347	2019-04-21 21:41:29.711347
2	2019-04-21 23:07:29.711347	2019-04-21 23:23:29.711347	2019-04-21 22:51:29.711347
3	2019-04-22 01:00:29.711347	2019-04-22 01:19:29.711347	2019-04-21 23:23:29.711347
4	2019-04-22 02:14:29.711347	2019-04-22 02:20:29.711347	2019-04-22 01:19:29.711347
5	2019-04-22 03:45:29.711347	2019-04-22 03:55:29.711347	2019-04-22 02:20:29.711347

```
# get rows where next admission is elective and replace with naT or nan
rows = df_adm.NEXT_ADMISSION_TYPE == 'ELECTIVE'
df_adm.loc[rows,'NEXT_ADMITTIME'] = pd.NaT
df_adm.loc[rows,'NEXT_ADMISSION_TYPE'] = np.NaN
```

```
# verify that it did what we wanted
df_adm.loc[df_adm.SUBJECT_ID == 124,['SUBJECT_ID','ADMITTIME','ADMISSION_TYPE','NEXT_ADMITTIME','NEXT_ADMISSION_TYPE']]
```

	SUBJECT_ID	ADMITTIME	ADMISSION_TYPE	NEXT_ADMITTIME	NEXT_ADMISSION_TYPE
165	124	2160-06-24 21:25:00	EMERGENCY	2161-12-17 03:39:00	EMERGENCY
166	124	2161-12-17 03:39:00	EMERGENCY	NaT	NaN
167	124	2165-05-21 21:02:00	ELECTIVE	2165-12-31 18:55:00	EMERGENCY
168	124	2165-12-31 18:55:00	EMERGENCY	NaT	NaN

Backward fill vs. Forward fill



```
[15] # sort by subject_ID and admission date
     # it is safer to sort right before the fill incase something changed the order above
     df_adm = df_adm.sort_values(['SUBJECT_ID','ADMITTIME'])

     # back fill (this will take a little while)
     df_adm[['NEXT_ADMITTIME','NEXT_ADMISSION_TYPE']] = df_adm.groupby(['SUBJECT_ID'])[['NEXT_ADMITTIME','NEXT_ADMISSION_TYPE']].fillna(method = 'bfill')
```

Backward fill

```
# importing pandas as pd
import pandas as pd

# Creating a dataframe with "na" values.

df = pd.DataFrame({"A":[None, 1, 2, 3, None, None],
                   "B":[11, 5, None, None, None, 8],
                   "C":[None, 5, 10, 11, None, 8]})

# Printing the dataframe
df
```

Original

	A	B	C
0	NaN	11.0	NaN
1	1.0	5.0	5.0
2	2.0	NaN	10.0
3	3.0	NaN	11.0
4	NaN	NaN	NaN
5	NaN	8.0	8.0

```
# Fill across the row
df.bfill(axis ='rows')
```

	A	B	C
0	1.0	11.0	5.0
1	1.0	5.0	5.0
2	2.0	8.0	10.0
3	3.0	8.0	11.0
4	NaN	8.0	8.0
5	NaN	8.0	8.0

```
# bfill values using values from next column
df.bfill(axis ='columns')
```

	A	B	C
0	11.0	11.0	NaN
1	1.0	5.0	5.0
2	2.0	10.0	10.0
3	3.0	11.0	11.0
4	NaN	NaN	NaN
5	8.0	8.0	8.0

Forward fill

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df=pd.DataFrame({"A":[5,3,None,4],
                 "B":[None,2,4,3],
                 "C":[4,3,8,5],
                 "D":[5,4,2,None]})

# Print the dataframe
df
```

Original

	A	B	C	D
0	5.0	NaN	4	5.0
1	3.0	2.0	3	4.0
2	NaN	4.0	8	2.0
3	4.0	3.0	5	NaN

`df.ffill(axis = 0)`

	A	B	C	D
0	5.0	NaN	4	5.0
1	3.0	2.0	3	4.0
2	3.0	4.0	8	2.0
3	4.0	3.0	5	2.0

`df.ffill(axis = 1)`

	A	B	C	D
0	5.0	5.0	4.0	5.0
1	3.0	2.0	3.0	4.0
2	NaN	4.0	8.0	2.0
3	4.0	3.0	5.0	5.0

Forward fill

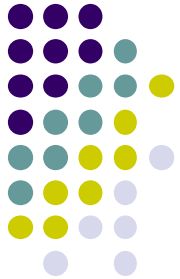
```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df=pd.DataFrame({"A":[5,3,None,4],
                 "B":[None,2,4,3],
                 "C":[4,3,8,5],
                 "D":[5,4,2,None]})

# Print the dataframe
df
```

Original

	A	B	C	D
0	5.0	NaN	4	5.0
1	3.0	2.0	3	4.0
2	NaN	4.0	8	2.0
3	4.0	3.0	5	NaN

```
df.ffill(axis = 0)
```

	A	B	C	D
0	5.0	NaN	4	5.0
1	3.0	2.0	3	4.0
2	3.0	4.0	8	2.0
3	4.0	3.0	5	2.0

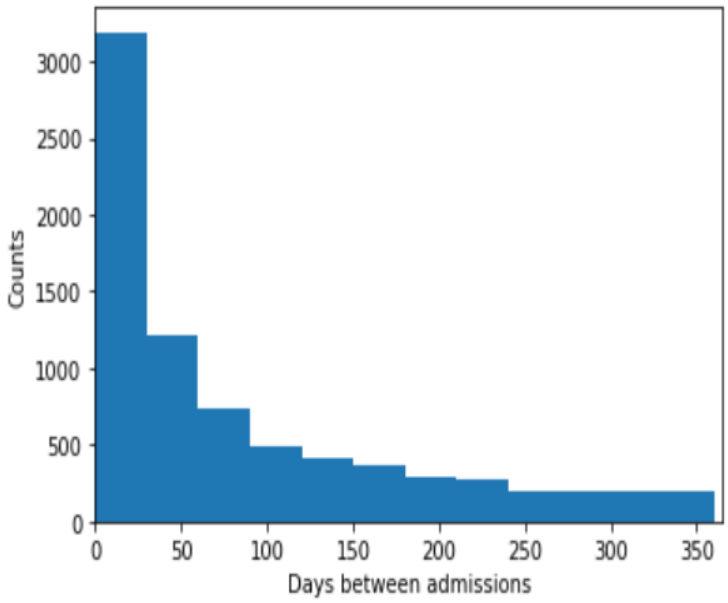


```
# verify that it did what we wanted
df_adm.loc[df_adm.SUBJECT_ID == 124,['SUBJECT_ID','ADMITTIME','ADMISSION_TYPE','NEXT_ADMITTIME','NEXT_ADMISSION_TYPE']]
```

	SUBJECT_ID	ADMITTIME	ADMISSION_TYPE	NEXT_ADMITTIME	NEXT_ADMISSION_TYPE
165	124	2160-06-24 21:25:00	EMERGENCY	2161-12-17 03:39:00	EMERGENCY
166	124	2161-12-17 03:39:00	EMERGENCY	2165-12-31 18:55:00	EMERGENCY
167	124	2165-05-21 21:02:00	ELECTIVE	2165-12-31 18:55:00	EMERGENCY
168	124	2165-12-31 18:55:00	EMERGENCY	NaT	NaN

```
[28] # calculate the number of days between discharge and next admission
     df_adm['DAYS_NEXT_ADMIT']=  (df_adm.NEXT_ADMITTIME - df_adm.DISCHTIME).dt.total_seconds()/(24*60*60)
```

```
# plot a histogram of days between readmissions if they exist
# this only works for non-null values so you have to filter
plt.hist(df_adm.loc[~df_adm.DAYS_NEXT_ADMIT.isnull(),'DAYS_NEXT_ADMIT'], bins =range(0,365,30))
plt.xlim([0,365])
plt.xlabel('Days between admissions')
plt.ylabel('Counts')
plt.show()
```



```
print('Number with a readmission:', (~df_adm.DAYS_NEXT_ADMIT.isnull()).sum())
print('Total Number:', len(df_adm))
```

```
Number with a readmission: 11399
Total Number: 58976
```

```
# verify that it did what we wanted
df_adm.loc[df_adm.SUBJECT_ID == 124,['SUBJECT_ID','ADMITTIME','ADMISSION_TYPE','DISCHTIME','NEXT_ADMITTIME','NEXT_ADMISSION_TYPE', 'DAYS_
```

	SUBJECT_ID	ADMITTIME	ADMISSION_TYPE	DISCHTIME	NEXT_ADMITTIME	NEXT_ADMISSION_TYPE	DAYS_NEXT_ADMIT
165	124	2160-06-24 21:25:00	EMERGENCY	2160-07-15 15:10:00	2161-12-17 03:39:00	EMERGENCY	519.520139
166	124	2161-12-17 03:39:00	EMERGENCY	2161-12-24 15:35:00	2165-12-31 18:55:00	EMERGENCY	1468.138889
167	124	2165-05-21 21:02:00	ELECTIVE	2165-06-06 16:00:00	2165-12-31 18:55:00	EMERGENCY	208.121528
168	124	2165-12-31 18:55:00	EMERGENCY	2166-02-01 06:55:00	NaT	NaN	NaN

Noteevents.csv

CATEGORY	count(*)
Nursing/other	822497
Radiology	522279
Nursing	223556
ECG	209051
Physician	141624
Discharge summary	59652
Echo	45794
Respiratory	31739
Nutrition	9418
General	8301
Rehab Services	5431
Social Work	2670
Case Management	967
Pharmacy	103
Consult	98

Load the notes

```
[ ]  from google.colab import files
     uploaded = files.upload()
```

Choose Files | No file chosen | Cancel upload

Saving NOTEEVENTS.csv.gz to NOTEEVENTS.csv.gz

```
df_notes = pd.read_csv("NOTEEVENTS.csv.gz")
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (4,5) have mixed types. Spe
  interactivity=interactivity, compiler=compiler, result=result)
```

```
print('Number of notes:',len(df_notes))
```

```
Number of notes: 2083180
```

```
df_notes.CATEGORY.unique()
```

```
array(['Discharge summary', 'Echo', 'ECG', 'Nursing', 'Physician ',
       'Rehab Services', 'Case Management ', 'Respiratory ', 'Nutrition',
       'General', 'Social Work', 'Pharmacy', 'Consult', 'Radiology',
       'Nursing/other'], dtype=object)
```

```
df_notes.head()
```

	ROW_ID	SUBJECT_ID	HADM_ID	CHARTDATE	CHARTTIME	STORETIME	CATEGORY	DESCRIPTION	CGID	ISERROR	TEXT
0	174	22532	167853.0	2151-08-04	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2151-7-16**] Dischar...
1	175	13702	107527.0	2118-06-14	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2118-6-2**] Discharg...
2	176	13702	167118.0	2119-05-25	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2119-5-4**] D...
3	177	13702	196489.0	2124-08-18	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2124-7-21**] ...
4	178	26880	135453.0	2162-03-25	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2162-3-3**] D...

```
# look at the first note
df_notes.TEXT.iloc[0]
```

```
'Admission Date:  [**2151-7-16**]       Discharge Date:  [**2151-8-4**]\n\n\nService:\nADDENDUM:\n\nRADIOLOGIC STUDIES:  Radiologic studies also incl
```

We can see that the dates and PHI have been converted for confidentiality. There are '\n' characters, numbers and punctuation.

At this point, we have to make a choice on what notes to use. For simplicity, let's use the discharge summary, but we could use all the notes by concatenating them.

```
# filter to discharge summary
df_notes_dis_sum = df_notes.loc[df_notes.CATEGORY == 'Discharge summary']
```

```
assert df_notes_dis_sum.duplicated(['HADM_ID']).sum() == 0, 'Multiple discharge summaries per admission'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-16-7553f51059ae> in <module>()
----> 1 assert df_notes_dis_sum.duplicated(['HADM_ID']).sum() == 0, 'Multiple discharge summaries per admi
ssion'

AssertionError: Multiple discharge summaries per admission
```

Because we have more than one discharge summary for one patient, we will take the last discharge summary.

```
df_notes_dis_sum_last = (df_notes_dis_sum.groupby(['SUBJECT_ID','HADM_ID']).nth(-1)).reset_index()
assert df_notes_dis_sum_last.duplicated(['HADM_ID']).sum() == 0, 'Multiple discharge summaries per admission'
```

```
df_adm_notes = pd.merge(df_adm[['SUBJECT_ID','HADM_ID','ADMITTIME','DISCHTIME','DAYS_NEXT_ADMIT','NEXT_ADMITTIME','ADMISSION_TYPE','DEATHTIME']],
                        df_notes_dis_sum_last[['SUBJECT_ID','HADM_ID','TEXT']],
                        on = ['SUBJECT_ID','HADM_ID'],
                        how = 'left')
assert len(df_adm) == len(df_adm_notes), 'Number of rows increased'
```

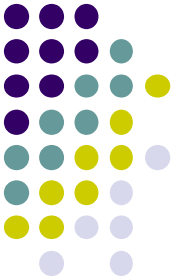
```
print('Fraction of missing notes:', df_adm_notes.TEXT.isnull().sum() / len(df_adm_notes))
print('Fraction notes with newlines:', df_adm_notes.TEXT.str.contains('\n').sum() / len(df_adm_notes))
print('Fraction notes with carriage returns:', df_adm_notes.TEXT.str.contains('\r').sum() / len(df_adm_notes))
```

```
Fraction of missing notes: 0.1059753119913185
Fraction notes with newlines: 0.8940246880086815
Fraction notes with carriage returns: 0.0
```

```
df_adm_notes.groupby('ADMISSION_TYPE').apply(lambda g: g.TEXT.isnull().sum())/df_adm_notes.groupby('ADMISSION_TYPE').size()
```

```
ADMISSION_TYPE
ELECTIVE     0.048663
EMERGENCY    0.037983
NEWBORN      0.536691
URGENT       0.042665
dtype: float64
```

Remove NEWBORN

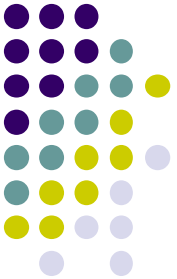


```
[34]
df_adm_notes_clean = df_adm_notes.loc[df_adm_notes.ADMISSION_TYPE != 'NEWBORN'].copy()
```

```
print('Fraction of missing notes:', df_adm_notes_clean.TEXT.isnull().sum() / len(df_adm_notes_clean))
print('Fraction notes with newlines:', df_adm_notes_clean.TEXT.str.contains('\n').sum() / len(df_adm_notes_clean))
print('Fraction notes with carriage returns:', df_adm_notes_clean.TEXT.str.contains('\r').sum() / len(df_adm_notes_clean))
```

```
Fraction of missing notes: 0.03971592354195606
Fraction notes with newlines: 0.9602840764580439
Fraction notes with carriage returns: 0.0
```

Prepare a label



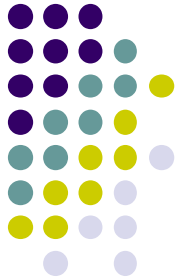
I like to create a specific column in the dataframe as OUTPUT_LABEL that has exactly what we are trying to predict. Here we want if the patient was re-admitted within 30 days

```
df_adm_notes_clean['OUTPUT_LABEL'] = (df_adm_notes_clean.DAYS_NEXT_ADMIT < 30).astype('int')
```

```
print('Number of positive samples:', (df_adm_notes_clean.OUTPUT_LABEL == 1).sum())
print('Number of negative samples:',  (df_adm_notes_clean.OUTPUT_LABEL == 0).sum())
print('Total samples:', len(df_adm_notes_clean))
```

```
Number of positive samples: 3004
Number of negative samples: 48109
Total samples: 51113
```

Create training and test dataframes

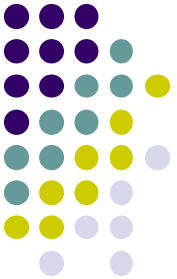


- When we build a predictive model, we want the model to work well on data that the model has never seen. To test for this, we take our data and split it into three datasets: training, validation and test.
- training set: used to train the model
- validation set: data the model didn't see, but are used to optimize or tune the model
- test set: data the model and tuning process never saw (true test of generalizability)
- The validation and test set should be as close to the production data as possible. We don't want to make decisions on validation data that is not from same type of data as the test set. For example, don't use high resolution images for validation set if you think production will be amateur photos from your phone.

```
# shuffle the samples
df_adm_notes_clean = df_adm_notes_clean.sample(n = len(df_adm_notes_clean), random_state = 42)
df_adm_notes_clean = df_adm_notes_clean.reset_index(drop = True)

# Save 30% of the data as validation and test data
df_valid_test=df_adm_notes_clean.sample(frac=0.30,random_state=42)

df_test = df_valid_test.sample(frac = 0.5, random_state = 42)
df_valid = df_valid_test.drop(df_test.index)

# use the rest of the data as training data
df_train_all=df_adm_notes_clean.drop(df_valid_test.index)

print('Test prevalence(n = %d):'%len(df_test),df_test.OUTPUT_LABEL.sum()/ len(df_test))
print('Valid prevalence(n = %d):'%len(df_valid),df_valid.OUTPUT_LABEL.sum()/ len(df_valid))
print('Train all prevalence(n = %d):'%len(df_train_all), df_train_all.OUTPUT_LABEL.sum()/ len(df_train_all))
print('all samples (n = %d)'%len(df_adm_notes_clean))
assert len(df_adm_notes_clean) == (len(df_test)+len(df_valid)+len(df_train_all)),'math didnt work'
```

```
Test prevalence(n = 7667): 0.061953828094430674
Valid prevalence(n = 7667): 0.056997521846876224
Train all prevalence(n = 35779): 0.05847005226529529
all samples (n = 51113)
```

Unbalanced data

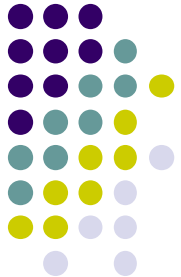


- Since the prevalence is so low, we want to prevent the model from always predicting negative. To do this, we have a few options
- balance the data by sub-sampling the negatives
- balance the data by over-sampling the positives
- create synthetic data (e.g. SMOTE)
- In this example, we will sub-sample the negatives

```
# split the training data into positive and negative
rows_pos = df_train_all.OUTPUT_LABEL == 1
df_train_pos = df_train_all.loc[rows_pos]
df_train_neg = df_train_all.loc[~rows_pos]

# merge the balanced data
df_train = pd.concat([df_train_pos, df_train_neg.sample(n = len(df_train_pos), random_state = 42)],axis = 0)

# shuffle the order of training samples
df_train = df_train.sample(n = len(df_train), random_state = 42).reset_index(drop = True)

print('Train prevalence (n = %d):'%len(df_train), df_train.OUTPUT_LABEL.sum()/ len(df_train))
```

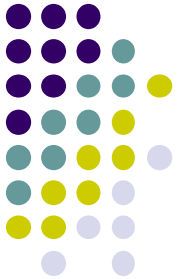
```
Train prevalence (n = 4184): 0.5
```

Preprocess text data



- Now that we have created data sets that have a label and the notes, we need to preprocess our text data to convert it to something useful (i.e. numbers) for the machine learning model. We are going to use the Bag-of-Words (BOW) approach.
- BOW basically breaks up the note into the individual words and counts how many times each word occurs. Your numerical data then becomes counts for some set of words as shown below. BOW is the simplest way to do NLP classification. In most blog posts I have read, fancier techniques have a hard time beating BOW for NLP classification tasks.
- In this process, there are few choices that need to be made
  - how to pre-process the words
  - how to count the words
  - which words to use
- There is no optimal choice for all NLP projects, so I recommend trying out a few options when building your own models.
- You can do the pre-processing in two ways:
- modify the original dataframe TEXT column
- pre-process as part of your pipeline so you don't edit the original data (allows you try out a few options easily)

Preprocess all the notes

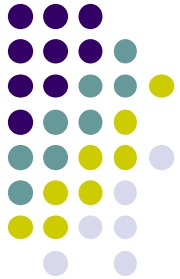


[40]

```
def preprocess_text(df):
    # This function preprocesses the text by filling not a number and replacing new lines ('\n') and carriage returns ('\r')
    df.TEXT = df.TEXT.fillna(' ')
    df.TEXT =df.TEXT.str.replace('\n',' ')
    df.TEXT =df.TEXT.str.replace('\r',' ')
    return df
```

```
# preprocess the text to deal with known issues
df_train = preprocess_text(df_train)
df_valid = preprocess_text(df_valid)
df_test = preprocess_text(df_test)
```

NLTK



NLTK 3.4.5 documentation

NEXT | MODULES | INDEX

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

NLTK Book

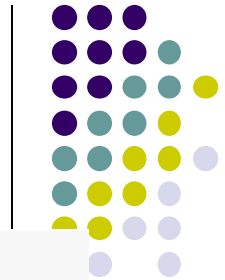
Natural Language Processing with Python

– Analyzing Text with the Natural Language Toolkit

Steven Bird, Ewan Klein, and Edward Loper

This version of the NLTK book is updated for Python 3 and NLTK 3. The first edition of the book, published by O'Reilly, is available at http://nltk.org/book_1ed/. (second edition of the book.)

https://www.nltk.org/book/



```
[ ]  nltk.download('all')
```

```
# Import PyDrive and associated libraries.
# This only needs to be done once per notebook.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

Build a tokenizer

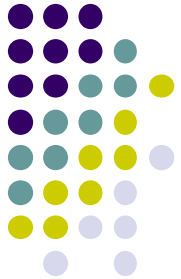
- Now we need to create a function that splits the note into individual words. This function is called a tokenizer. We can use the nltk.word_tokenize function to this for us.

```
nltk.download('all')
```

```
import nltk
from nltk import word_tokenize
word_tokenize('This should be tokenized. 02/02/2018 sentence has stars**')
```

```
['This',
 'should',
 'be',
 'tokenized',
 '.',
 '02/02/2018',
 'sentence',
 'has',
 'stars**']
```

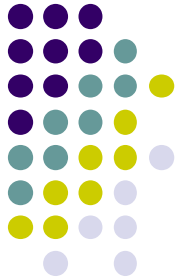
New tokenizer



- Note that some punctuation is separated but not all punctuation.
- We can define our own custom tokenizer. Our tokenizer will
- replace punctuation with spaces
- remove numbers with spaces
- lowercase all words
- I like to do these steps as part of the tokenizer, so I can go back and look at the original note. The new line and carriage return is more difficult to deal with so I did it in the preprocessing step above.

```
import string
print(string.punctuation)
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```



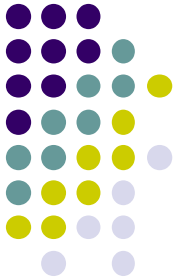
[46]

```
def tokenizer_better(text):
    # tokenize the text by replacing punctuation and numbers with spaces and lowercase all words

    punc_list = string.punctuation+'0123456789'
    t = str.maketrans(dict.fromkeys(punc_list, " "))
    text = text.lower().translate(t)
    tokens = word_tokenize(text)
    return tokens
```

```
tokenizer_better('This should be tokenized. 02/02/2018 sentence has stars**')
```

```
['this', 'should', 'be', 'tokenized', 'sentence', 'has', 'stars']
```

Build a simple vectorizer



```
[48] sample_text = ['Data science is about the data', 'The science is amazing', 'Predictive modeling is part of data science']
```

```
[49] from sklearn.feature_extraction.text import CountVectorizer
     vect = CountVectorizer(tokenizer = tokenizer_better)
     vect.fit(sample_text)

     # matrix is stored as a sparse matrix (since you have a lot of zeros
     X = vect.transform(sample_text)
```

```
X
```

```
<3x10 sparse matrix of type '<class 'numpy.int64'>'
        with 16 stored elements in Compressed Sparse Row format>
```

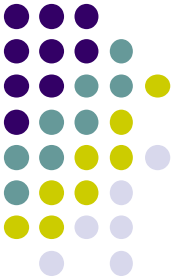
```
[51] # we can visualize this small example if we convert it to an array
     X.toarray()
```

```
array([[1, 0, 2, 1, 0, 0, 0, 0, 1, 1],
       [0, 1, 0, 1, 0, 0, 0, 0, 1, 1],
       [0, 0, 1, 1, 1, 1, 1, 1, 1, 0]])
```

```
# get the column names
vect.get_feature_names()
```

```
['about',
 'amazing',
 'data',
 'is',
 'modeling',
 'of',
 'part',
 'predictive',
 'science',
 'the']
```

build a vectorizer on the clinical notes

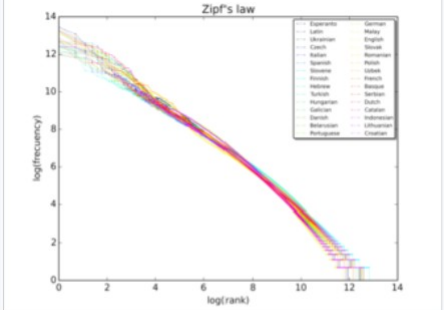


```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(max_features = 3000, tokenizer = tokenizer_better)

# this could take a while
vect.fit(df_train.TEXT.values)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=3000, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=<function tokenizer_better at 0x7f63ccbca158>,
                vocabulary=None)
```

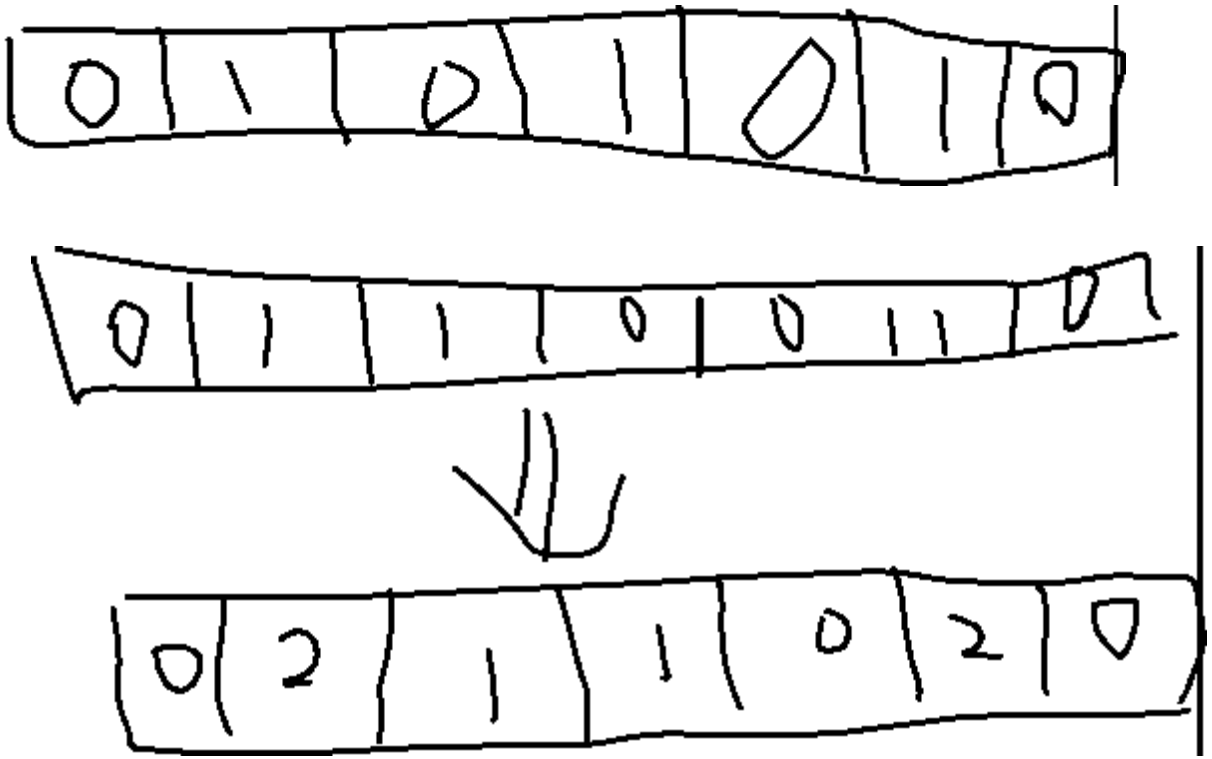
Zipf's law

```
neg_doc_matrix = vect.transform(df_train[df_train.OUTPUT_LABEL == 0].TEXT)
pos_doc_matrix = vect.transform(df_train[df_train.OUTPUT_LABEL == 1].TEXT)
neg_tf = np.sum(neg_doc_matrix,axis=0)
pos_tf = np.sum(pos_doc_matrix,axis=0)
neg = np.squeeze(np.asarray(neg_tf))
pos = np.squeeze(np.asarray(pos_tf))

term_freq_df = pd.DataFrame([neg,pos],columns=vect.get_feature_names()).transpose()
term_freq_df.columns = ['negative', 'positive']
term_freq_df['total'] = term_freq_df['negative'] + term_freq_df['positive']
term_freq_df.sort_values(by='total', ascending=False).iloc[:10]

#Create a series from the sparse matrix
d = pd.Series(term_freq_df.total,
              index = term_freq_df.index).sort_values(ascending=False)
ax = d[:50].plot(kind='bar', figsize=(10,6), width=.8, fontsize=14, rot=90,color = 'b')
ax.title.set_size(18)
plt.ylabel('count')
plt.show()
ax = d[50:100].plot(kind='bar', figsize=(10,6), width=.8, fontsize=14, rot=90,color = 'b')
ax.title.set_size(18)
plt.ylabel('count')
plt.show()
```



Zipf's law plot for the first 10 million words in 30 Wikipedias (as of October 2015) in a log-log scale.

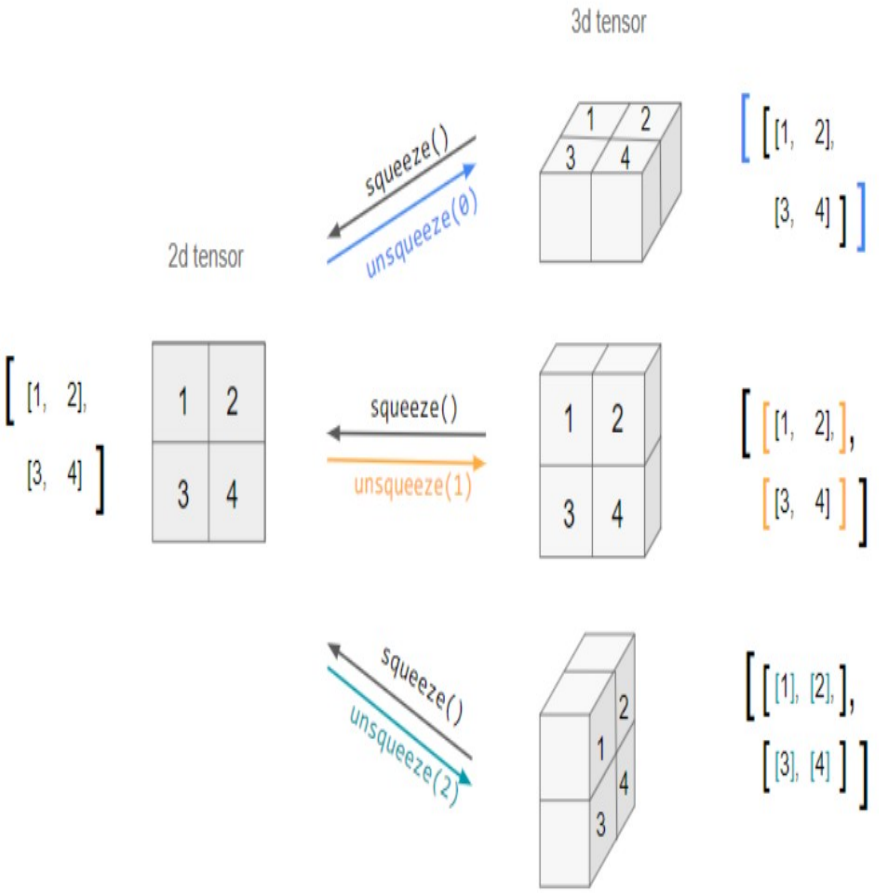
Sum()



Squeeze()

```
Input array :  [[[12 42 22]
  [ 9 12  2]]]
output squeezed array :  [[12 42 22]
[ 9 12  2]]
```

```
Input array :  [[[0 1]
  [2 3]
  [4 5]
  [6 7]]]
output array :  [[0 1]
[2 3]
[4 5]
[6 7]]
```

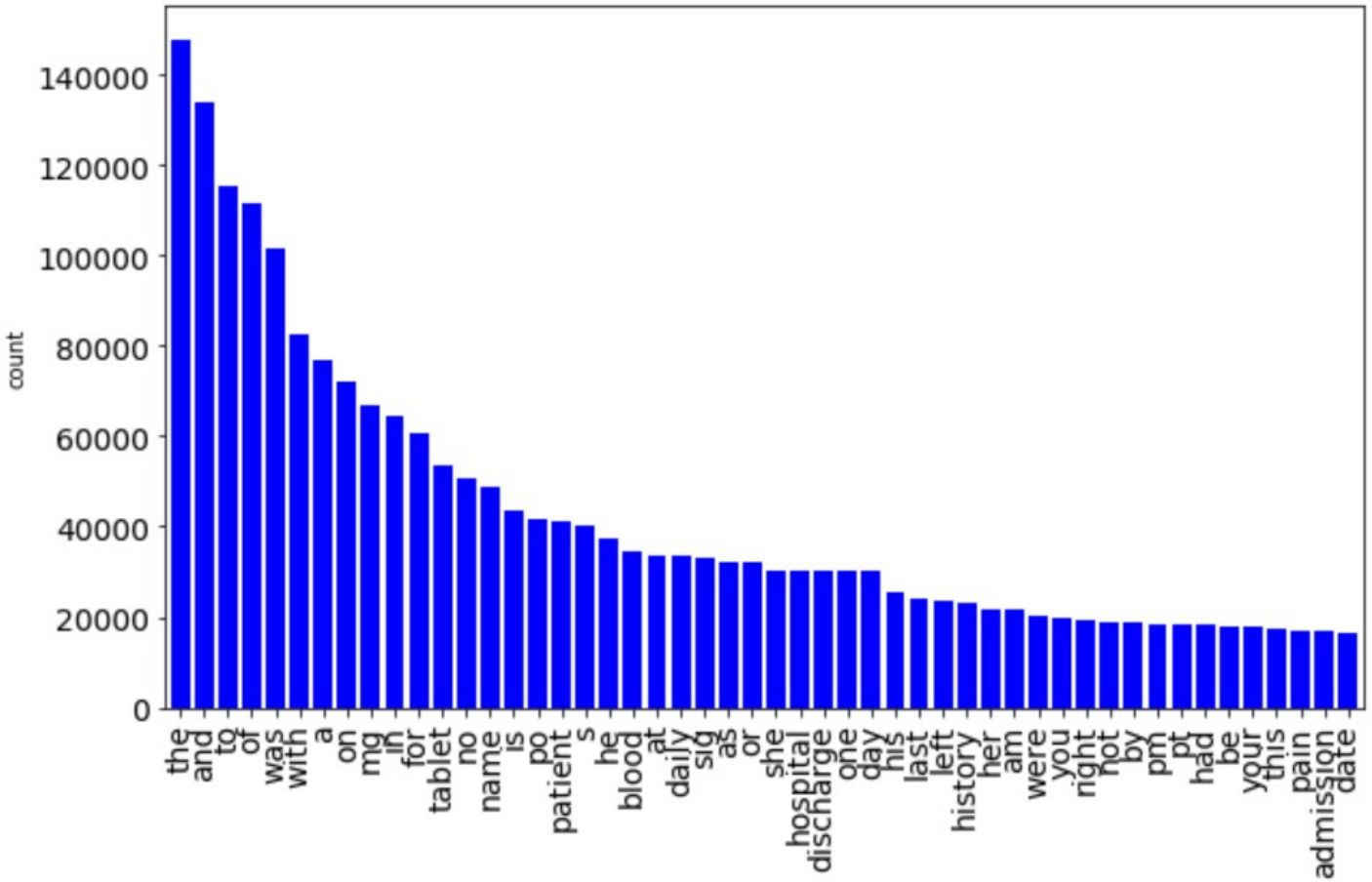


https://www.tutorialandexample.com/numpy-squeeze-in-python

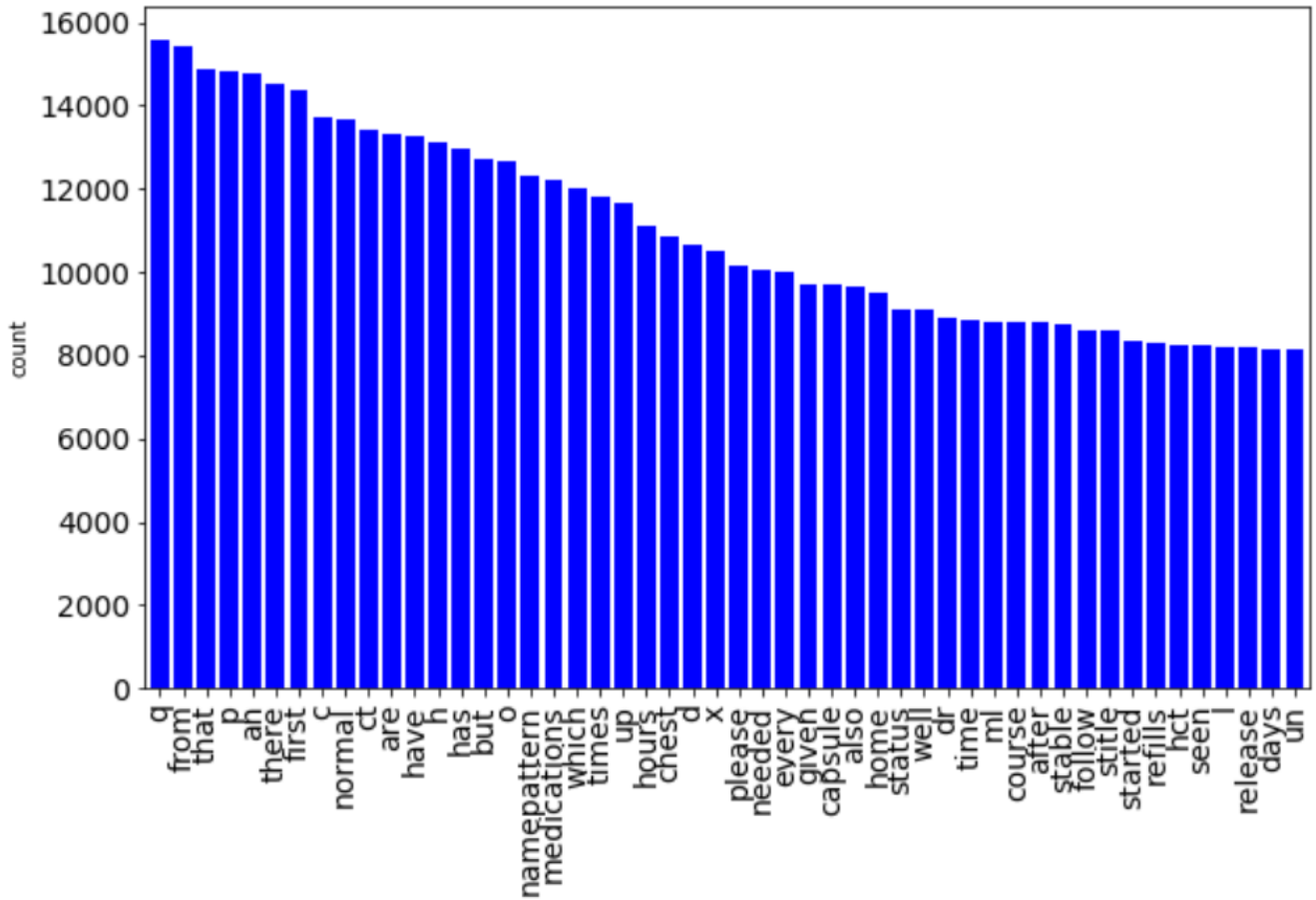
Vector transpose

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}$$

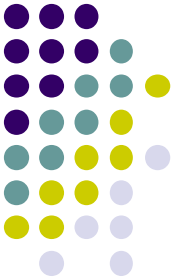
Top 50 words


count
140000
120000
100000
80000
60000
40000
20000
0
the
and
to
of
was
with
a
on
mg
in
for
tablet
no
name
is
po
patient
s
he
blood
at
daily
sig
as
or
she
hospital
discharge
one
day
his
last
left
history
her
am
were
you
night
not
by
pm
pt
had
be
your
this
pain
admission
date

To 51-100 words



Adding stop-words

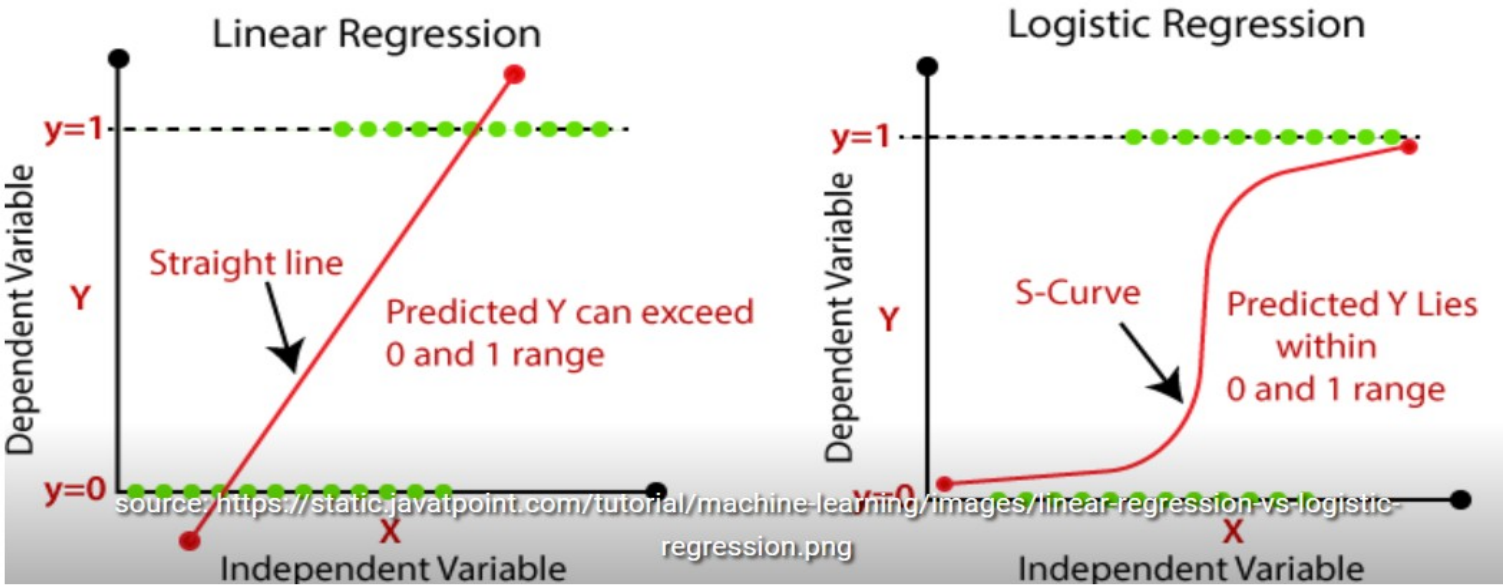


```
my_stop_words = ['the','and','to','of','was','with','a','on','in','for','name',
                 'is','patient','s','he','at','as','or','one','she','his','her','am',
                 'were','you','pt','pm','by','be','had','your','this','date',
                'from','there','an','that','p','are','have','has','h','but','o',
                'namepattern','which','every','also']
```

```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(max_features = 3000,
                       tokenizer = tokenizer_better,
                       stop_words = my_stop_words)
# this could take a while
vect.fit(df_train.TEXT.values)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=3000, min_df=1,
                ngram_range=(1, 1), preprocessor=None,
                stop_words=['the', 'and', 'to', 'of', 'was', 'with', 'a', 'on',
                            'in', 'for', 'name', 'is', 'patient', 's', 'he',
                            'at', 'as', 'or', 'one', 'she', 'his', 'her', 'am',
                            'were', 'you', 'pt', 'pm', 'by', 'be', 'had', ...],
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=<function tokenizer_better at 0x7f63ccbca158>,
                vocabulary=None)
```

Logistic Regression



source: https://static.javatpoint.com/tutorial/machine-learning/images/linear-regression-vs-logistic-regression.png

Build a simple predictive model

```
# logistic regression
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(C = 0.0001, penalty = 'l2', random_state = 42)
clf.fit(X_train_tf, y_train)
```

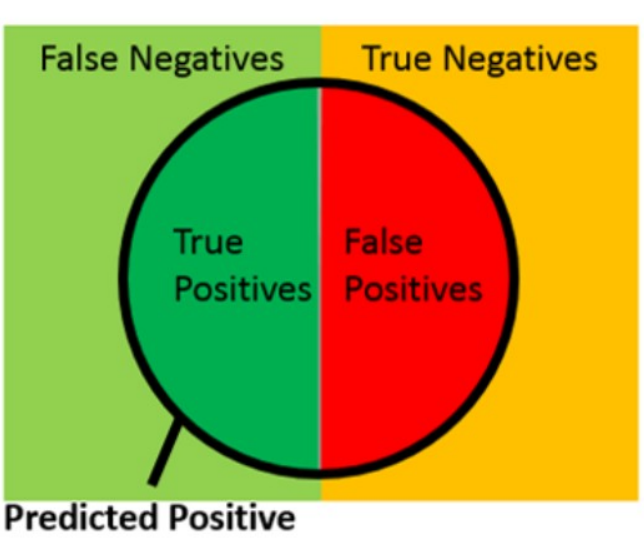
```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWarning:
  FutureWarning)
LogisticRegression(C=0.0001, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
[60] model = clf
     y_train_preds = model.predict_proba(X_train_tf)[:,1]
     y_valid_preds = model.predict_proba(X_valid_tf)[:,1]
```

```
print(y_train[:10].values)
print(y_train_preds[:10])
```

```
[1 1 0 1 1 1 0 0 1 1]
[0.4542033  0.46177468 0.29772094 0.6233307  0.24774543 0.47941329
 0.36292154 0.90984735 0.69831955 0.50604926]
```

Calculate Performance Metrics



Prevalence =

Fraction of positives in population

Accuracy =

Fraction predicted correctly

Recall (Sensitivity) =

Fraction of positives predicted correctly

Specificity =

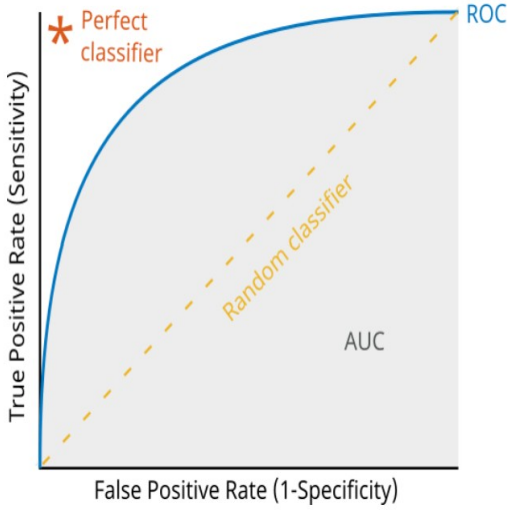
Fraction of negatives predicted correctly

Precision =

Fraction of predicted positives that are actually positive

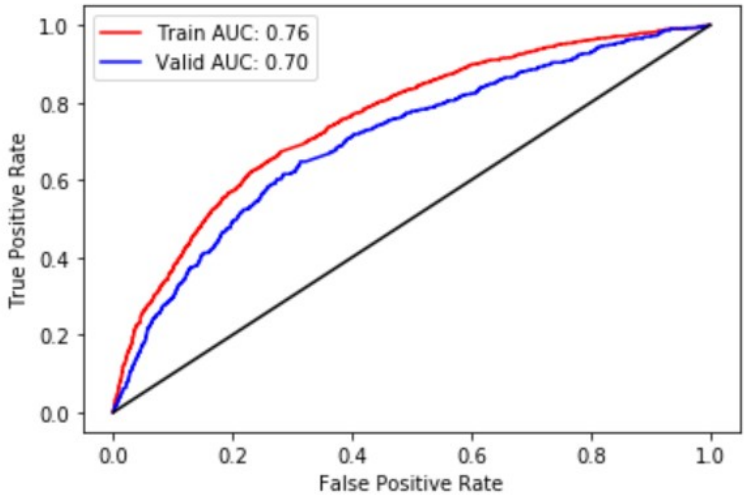
Example: Positive = Hospitalized, Negative = Not Hospitalized

```python
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_preds)
fpr_valid, tpr_valid, thresholds_valid = roc_curve(y_valid, y_valid_preds)


thresh = 0.5

auc_train = roc_auc_score(y_train, y_train_preds)
auc_valid = roc_auc_score(y_valid, y_valid_preds)

print('Train AUC:%.3f'%auc_train)
print('Valid AUC:%.3f'%auc_valid)

print('Train accuracy:%.3f'%calc_accuracy(y_train, y_train_preds, thresh))
print('Valid accuracy:%.3f'%calc_accuracy(y_valid, y_valid_preds, thresh))


print('Train recall:%.3f'%calc_recall(y_train, y_train_preds, thresh))
print('Valid recall:%.3f'%calc_recall(y_valid, y_valid_preds, thresh))

print('Train precision:%.3f'%calc_precision(y_train, y_train_preds, thresh))
print('Valid precision:%.3f'%calc_precision(y_valid, y_valid_preds, thresh))
```



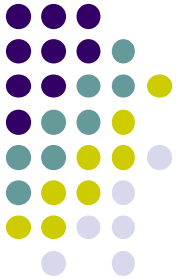
```
print('Train specificity:%.3f'%calc_specificity(y_train, y_train_preds, thresh))
print('Valid specificity:%.3f'%calc_specificity(y_valid, y_valid_preds, thresh))

print('Train prevalence:%.3f'%calc_prevalence(y_train))
print('Valid prevalence:%.3f'%calc_prevalence(y_valid))


plt.plot(fpr_train, tpr_train,'r-', label = 'Train AUC: %.2f'%auc_train)
plt.plot(fpr_valid, tpr_valid,'b-',label = 'Valid AUC: %.2f'%auc_valid)
plt.plot([0,1],[0,1],'-k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
Train AUC:0.757
Valid AUC:0.704
Train accuracy:0.695
Valid accuracy:0.682
Train recall:0.666
Valid recall:0.648
Train precision:0.706
Valid precision:0.110
Train specificity:0.723
Valid specificity:0.684
Train prevalence:0.500
Valid prevalence:0.057
```

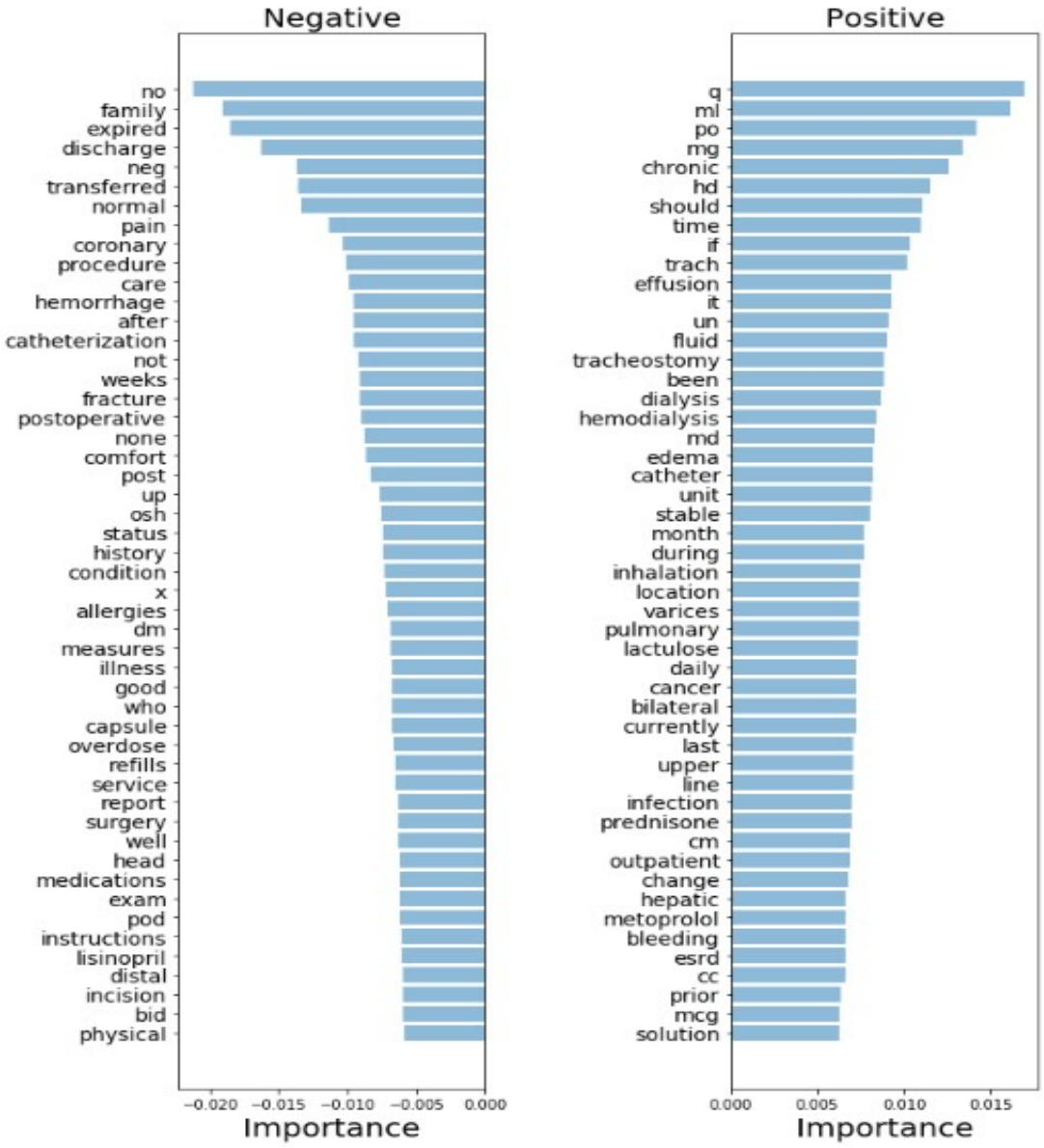


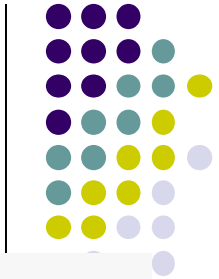
Try to improve the model

- Visualize the top words for positive and negative classes to see if there are any patterns which could give insight into additional features to add or remove



```
top_words = [a[0] for a in top_pairs]
top_scores = [a[1] for a in top_pairs]

bottom_words = [a[0] for a in bottom_pairs]
bottom_scores = [a[1] for a in bottom_pairs]

fig = plt.figure(figsize=(10, 15))

plt.subplot(121)
plt.barh(y_pos,bottom_scores, align='center', alpha=0.5)
plt.title('Negative', fontsize=20)
plt.yticks(y_pos, bottom_words, fontsize=14)
plt.suptitle('Key words', fontsize=16)
plt.xlabel('Importance', fontsize=20)

plt.subplot(122)
plt.barh(y_pos,top_scores, align='center', alpha=0.5)
plt.title('Positive', fontsize=20)
plt.yticks(y_pos, top_words, fontsize=14)
plt.suptitle(name, fontsize=16)
plt.xlabel('Importance', fontsize=20)

plt.subplots_adjust(wspace=0.8)
plt.show()
```


Negative
no
family
expired
discharge
neg
transferred
normal
pain
coronary
procedure
care
hemorrhage
after
catheterization
not
weeks
fracture
postoperative
none
comfort
post
up
osh
status
history
condition
x
allergies
dm
measures
illness
good
who
capsule
overdose
refills
service
report
surgery
well
head
medications
exam
pod
instructions
lisinopril
distal
incision
bid
physical
−0.020
−0.015
−0.010
−0.005
0.000
Importance
Positive
q
ml
po
mg
chronic
hd
should
time
if
trach
effusion
it
un
fluid
tracheostomy
been
dialysis
hemodialysis
md
edema
catheter
unit
stable
month
during
inhalation
location
varices
pulmonary
lactulose
daily
cancer
bilateral
currently
last
upper
line
infection
prednisone
cm
outpatient
change
hepatic
metoprolol
bleeding
esrd
cc
prior
mcg
solution
0.000
0.005
0.010
0.015
Importance



New stop words

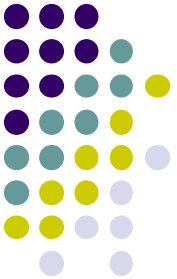
```
[ ]
    my_new_stop_words = ['the','and','to','of','was','with','a','on','in','for','name',
                         'is','patient','s','he','at','as','or','one','she','his','her','am',
                         'were','you','pt','pm','by','be','had','your','this','date',
                        'from','there','an','that','p','are','have','has','h','but','o',
                        'namepattern','which','every','also','should','if','it','been','who','during', 'x']
```

```
[ ]  vect = CountVectorizer(lowercase = True, max_features = 3000, tokenizer = tokenizer_better,stop_words =my_new_stop_words)
     #
     # This could take a while
     vect.fit(df_train.TEXT.values)

     X_train_tf = vect.transform(df_train.TEXT.values)
     X_valid_tf = vect.transform(df_valid.TEXT.values)
     y_train = df_train.OUTPUT_LABEL
     y_valid = df_valid.OUTPUT_LABEL
```

Previous stop words

```
my_stop_words = ['the','and','to','of','was','with','a','on','in','for','name',
                 'is','patient','s','he','at','as','or','one','she','his','her','am',
                 'were','you','pt','pm','by','be','had','your','this','date',
                'from','there','an','that','p','are','have','has','h','but','o',
                'namepattern','which','every','also']
```

plot a learning curve

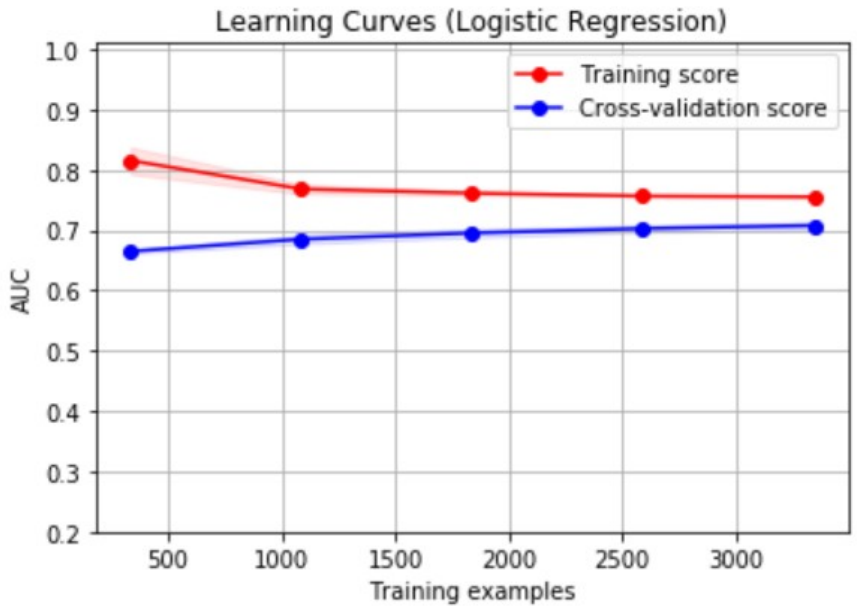


```
[ ]  import numpy as np
     from sklearn.model_selection import learning_curve
     from sklearn.model_selection import ShuffleSplit


     def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                             n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
         """
         Generate a simple plot of the test and training learning curve.

         Parameters
         ----------
         estimator : object type that implements the "fit" and "predict" methods
             An object of that type which is cloned for each validation.

         title : string
             Title for the chart.

         X : array-like, shape (n_samples, n_features)
             Training vector, where n_samples is the number of samples and
             n_features is the number of features.

         y : array-like, shape (n_samples) or (n_samples, n_features), optional
             Target relative to X for classification or regression;
             None for unsupervised learning.
```

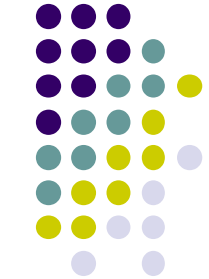
```
ylim : tuple, shape (ymin, ymax), optional
    Defines minimum and maximum yvalues plotted.

cv : int, cross-validation generator or an iterable, optional
    Determines the cross-validation splitting strategy.
    Possible inputs for cv are:
      - None, to use the default 3-fold cross-validation,
      - integer, to specify the number of folds.
      - An object to be used as a cross-validation generator.
      - An iterable yielding train/test splits.

    For integer/None inputs, if ``y`` is binary or multiclass,
    :class:`StratifiedKFold` used. If the estimator is not a classifier
    or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

    Refer :ref:`User Guide <cross_validation>` for the various
    cross-validators that can be used here.

n_jobs : integer, optional
    Number of jobs to run in parallel (default 1).
"""
```

```
plt.figure()
plt.title(title)
if ylim is not None:
    plt.ylim(*ylim)
plt.xlabel("Training examples")
plt.ylabel("AUC")
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring = 'roc_auc')
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="b")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="b",
         label="Cross-validation score")
```

```
[ ]
    title = "Learning Curves (Logistic Regression)"
    # Cross validation with 5 iterations to get smoother mean test and train
    # score curves, each time with 20% data randomly selected as a validation set.
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
    estimator = LogisticRegression( C = 0.0001, penalty = 'l2')
    plot_learning_curve(estimator, title, X_train_tf, y_train, ylim=(0.2, 1.01), cv=cv, n_jobs=4)

    plt.show()
```



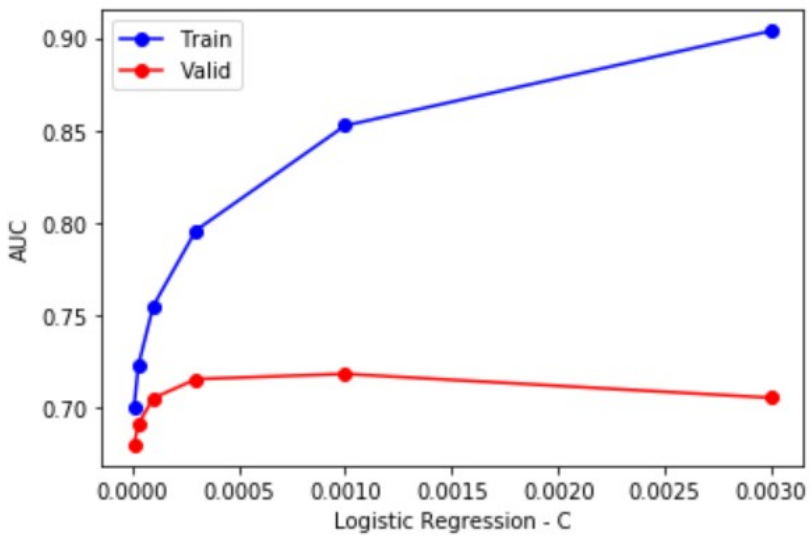
how the hyperparameters affect your result



```
from sklearn.linear_model import LogisticRegression

Cs = [0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003]
train_aucs = np.zeros(len(Cs))
valid_aucs = np.zeros(len(Cs))

for ii in range(len(Cs)):
    C = Cs[ii]
    print('\n C:', C)

    # logistic regression

    clf=LogisticRegression(C = C, penalty = 'l2', random_state = 42)
    clf.fit(X_train_tf, y_train)

    model = clf
    y_train_preds = model.predict_proba(X_train_tf)[:,1]
    y_valid_preds = model.predict_proba(X_valid_tf)[:,1]

    auc_train = roc_auc_score(y_train, y_train_preds)
    auc_valid = roc_auc_score(y_valid, y_valid_preds)
    print('Train AUC:%.3f'%auc_train)
    print('Valid AUC:%.3f'%auc_valid)
    train_aucs[ii] = auc_train
    valid_aucs[ii] = auc_valid
```

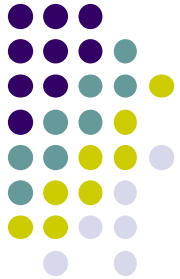
```
[ ]  C: 1e-05
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
  FutureWarning)
Train AUC:0.700
Valid AUC:0.680

 C: 3e-05
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
  FutureWarning)
Train AUC:0.723
Valid AUC:0.691

 C: 0.0001
/usr/local/lib/python3.6/dist-packages/sk
  FutureWarning)
Train AUC:0.755
Valid AUC:0.705

 C: 0.0003
/usr/local/lib/python3.6/dist-packages/sk
  FutureWarning)
Train AUC:0.796
Valid AUC:0.715

 C: 0.001
/usr/local/lib/python3.6/dist-packages/sk
  FutureWarning)
Train AUC:0.853
Valid AUC:0.718
```

```
[ ]
plt.plot(Cs, train_aucs,'bo-', label ='Train')
plt.plot(Cs, valid_aucs, 'ro-', label='Valid')
plt.legend()
plt.xlabel('Logistic Regression - C')
plt.ylabel('AUC')
plt.show()
```

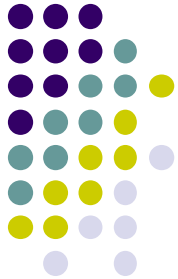


```
[ ]  num_features = [100,300,1000,3000,10000,30000]
     train_aucs = np.zeros(len(num_features))
     valid_aucs = np.zeros(len(num_features))

     for ii in range(len(num_features)):
         num = num_features[ii]
         print('\nnumber of features:', num)
         vect = CountVectorizer(lowercase = True, max_features = num,
                                tokenizer = tokenizer_better,stop_words =my_new_stop_words)

         # This could take a while
         vect.fit(df_train.TEXT.values)

         X_train_tf = vect.transform(df_train.TEXT.values)
         X_valid_tf = vect.transform(df_valid.TEXT.values)
         y_train = df_train.OUTPUT_LABEL
         y_valid = df_valid.OUTPUT_LABEL

         clf=LogisticRegression(C = 0.0001, penalty = 'l2', random_state = 42)
         clf.fit(X_train_tf, y_train)

         model = clf
         y_train_preds = model.predict_proba(X_train_tf)[:,1]
         y_valid_preds = model.predict_proba(X_valid_tf)[:,1]
```

Final model



```
[ ]  # shuffle the samples

     rows_not_death = df_adm_notes_clean.DEATHTIME.isnull()

     df_adm_notes_not_death = df_adm_notes_clean.loc[rows_not_death].copy()
     df_adm_notes_not_death = df_adm_notes_not_death.sample(n = len(df_adm_notes_not_death), random_state = 42)
     df_adm_notes_not_death = df_adm_notes_not_death.reset_index(drop = True)

     # Save 30% of the data as validation and test data
     df_valid_test=df_adm_notes_not_death.sample(frac=0.30,random_state=42)

     df_test = df_valid_test.sample(frac = 0.5, random_state = 42)
     df_valid = df_valid_test.drop(df_test.index)

     # use the rest of the data as training data
     df_train_all=df_adm_notes_not_death.drop(df_valid_test.index)

     print('Test prevalence(n = %d):'%len(df_test),df_test.OUTPUT_LABEL.sum()/ len(df_test))
     print('Valid prevalence(n = %d):'%len(df_valid),df_valid.OUTPUT_LABEL.sum()/ len(df_valid))
     print('Train all prevalence(n = %d):'%len(df_train_all), df_train_all.OUTPUT_LABEL.sum()/ len(df_train_all))
     print('all samples (n = %d)'%len(df_adm_notes_clean))
     assert len(df_adm_notes_not_death) == (len(df_test)+len(df_valid)+len(df_train_all)),'math didnt work'
```

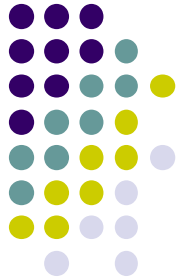


```
# split the training data into positive and negative
rows_pos = df_train_all.OUTPUT_LABEL == 1
df_train_pos = df_train_all.loc[rows_pos]
df_train_neg = df_train_all.loc[~rows_pos]

# merge the balanced data
df_train = pd.concat([df_train_pos, df_train_neg.sample(n = len(df_train_pos), random_state = 42)],axis = 0)

# shuffle the order of training samples
df_train = df_train.sample(n = len(df_train), random_state = 42).reset_index(drop = True)

print('Train prevalence (n = %d):'%len(df_train), df_train.OUTPUT_LABEL.sum()/ len(df_train))

# preprocess the text to deal with known issues
df_train = preprocess_text(df_train)
df_valid = preprocess_text(df_valid)
df_test = preprocess_text(df_test)
```

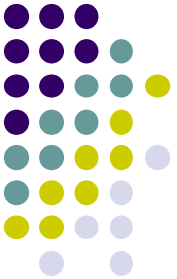
```
Test prevalence(n = 6798): 0.06590173580464842
Valid prevalence(n = 6798): 0.06913798175934098
Train all prevalence(n = 31725): 0.0644602048857368
all samples (n = 51113)
Train prevalence (n = 4090): 0.5
```

```
my_new_stop_words = ['the','and','to','of','was','with','a','on','in','for','name',
                     'is','patient','s','he','at','as','or','one','she','his','her','am',
                     'were','you','pt','pm','by','be','had','your','this','date',
                    'from','there','an','that','p','are','have','has','h','but','o',
                    'namepattern','which','every','also','should','if','it','been','who','during', 'x', 'q', 'no', 'ml', 'po',

from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(lowercase = True, max_features = 3000,
                       tokenizer = tokenizer_better,
                      stop_words = my_new_stop_words)

# This could take a while
vect.fit(df_train.TEXT.values)

X_train_tf = vect.transform(df_train.TEXT.values
X_valid_tf = vect.transform(df_valid.TEXT.values
X_test_tf = vect.transform(df_test.TEXT.values)

y_train = df_train.OUTPUT_LABEL
y_valid = df_valid.OUTPUT_LABEL
y_test = df_test.OUTPUT_LABEL

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

```
clf=LogisticRegression(C = 0.0003, penalty = 'l2', random_state = 42)
clf.fit(X_train_tf, y_train)

model = clf
y_train_preds = model.predict_proba(X_train_tf)[:,1]
y_valid_preds = model.predict_proba(X_valid_tf)[:,1]
y_test_preds = model.predict_proba(X_test_tf)[:,1]
```

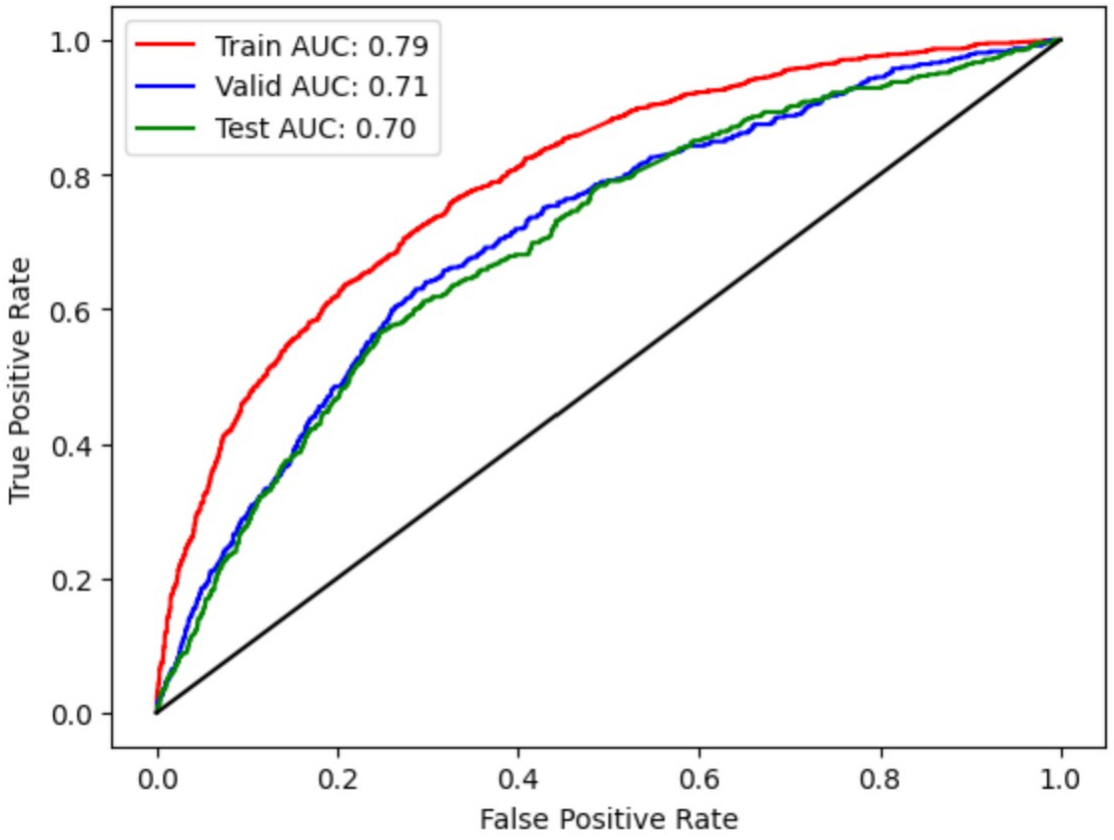


```
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_preds)
fpr_valid, tpr_valid, thresholds_valid = roc_curve(y_valid, y_valid_preds)
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_preds)

thresh = 0.5

auc_train = roc_auc_score(y_train, y_train_preds)
auc_valid = roc_auc_score(y_valid, y_valid_preds)
auc_test = roc_auc_score(y_test, y_test_preds)

print('Train prevalence(n = %d): %.3f'%(len(y_train),sum(y_train)/ len(y_train)))
print('Valid prevalence(n = %d): %.3f'%(len(y_valid),sum(y_valid)/ len(y_valid)))
print('Test prevalence(n = %d): %.3f'%(len(y_test),sum(y_test)/ len(y_test)))

print('Train AUC:%.3f'%auc_train)
print('Valid AUC:%.3f'%auc_valid)
print('Test AUC:%.3f'%auc_test)

print('Train accuracy:%.3f'%calc_accuracy(y_train, y_train_preds, thresh))
print('Valid accuracy:%.3f'%calc_accuracy(y_valid, y_valid_preds, thresh))
print('Test accuracy:%.3f'%calc_accuracy(y_test, y_test_preds, thresh))
```



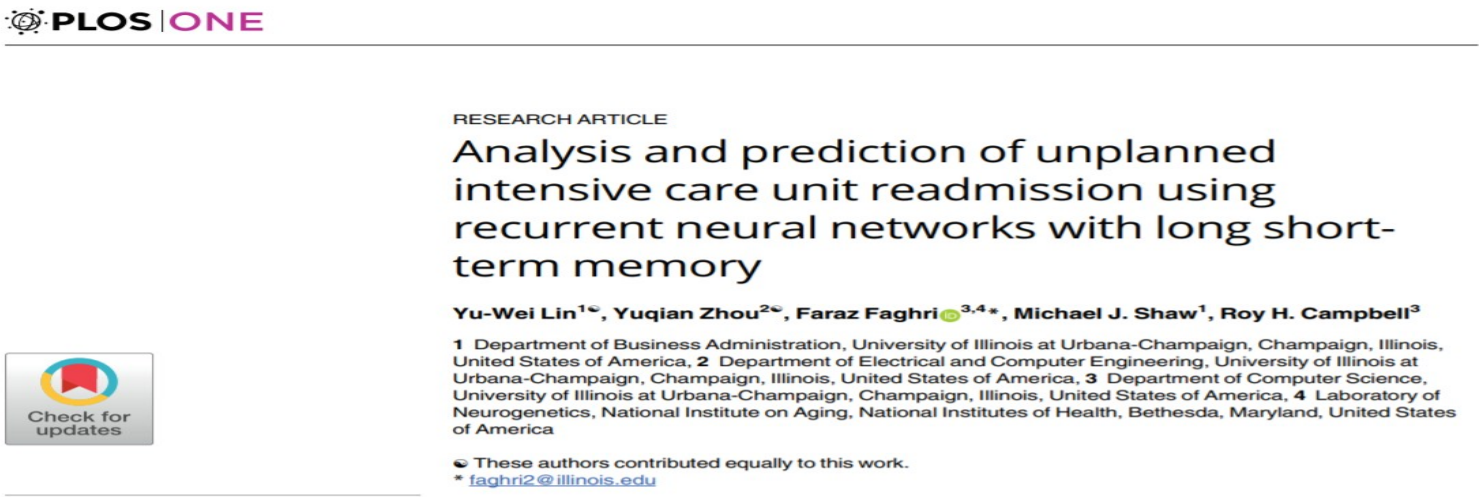
```
print('Train recall:%.3f'%calc_recall(y_train, y_train_preds, thresh))
print('Valid recall:%.3f'%calc_recall(y_valid, y_valid_preds, thresh))
print('Test recall:%.3f'%calc_recall(y_test, y_test_preds, thresh))

print('Train precision:%.3f'%calc_precision(y_train, y_train_preds, thresh))
print('Valid precision:%.3f'%calc_precision(y_valid, y_valid_preds, thresh))
print('Test precision:%.3f'%calc_precision(y_test, y_test_preds, thresh))

print('Train specificity:%.3f'%calc_specificity(y_train, y_train_preds, thresh))
print('Valid specificity:%.3f'%calc_specificity(y_valid, y_valid_preds, thresh))
print('Test specificity:%.3f'%calc_specificity(y_test, y_test_preds, thresh))

plt.plot(fpr_train, tpr_train,'r-', label = 'Train AUC: %.2f'%auc_train)
plt.plot(fpr_valid, tpr_valid,'b-',label = 'Valid AUC: %.2f'%auc_valid)
plt.plot(fpr_test, tpr_test,'g-',label = 'Test AUC: %.2f'%auc_test)

plt.plot([0,1],[0,1],'-k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
Train prevalence(n = 4090): 0.500
Valid prevalence(n = 6798): 0.069
Test prevalence(n = 6798): 0.066
Train AUC:0.793
Valid AUC:0.711
Test AUC:0.700
Train accuracy:0.710
Valid accuracy:0.701
Test accuracy:0.707
Train recall:0.660
Valid recall:0.630
Test recall:0.598
Train precision:0.733
Valid precision:0.137
Test precision:0.129
Train specificity:0.760
Valid specificity:0.706
Test specificity:0.714
```



More Machine Learning/Deep Learning methods

- Using MIMIC III ICU data



PLOS ONE

RESEARCH ARTICLE

Analysis and prediction of unplanned intensive care unit readmission using recurrent neural networks with long short-term memory

Yu-Wei Lin[1☯], Yuqian Zhou[2☯], Faraz Faghri[3,4*], Michael J. Shaw[1], Roy H. Campbell[3]

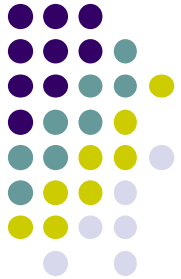
1 Department of Business Administration, University of Illinois at Urbana-Champaign, Champaign, Illinois, United States of America, **2** Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, Illinois, United States of America, **3** Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, Illinois, United States of America, **4** Laboratory of Neurogenetics, National Institute on Aging, National Institutes of Health, Bethesda, Maryland, United States of America

☯ These authors contributed equally to this work.
* faghri2@illinois.edu

Check for updates

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6613707/pdf/pone.0218942.pdf

https://github.com/Jeffreylin0925/MIMIC-III_ICU_Readmission_Analysis



Thank you!