Drug reviews sentiment analysis

Background information

A hospital or insurance provider is interested in efficiently extracting numeric ratings from patients' written review. To this end we build a model using labelled, numerically, patient reviews.

Data exploration

The data comes from Drugs.com and is accessed through UCI's website. Click here to access data. Data was sourced by Kalummadi and Grer.

In [1]:
```
import pandas as pd
```

In [2]:
```
import numpy as np
```

In [3]:
```
import matplotlib.pyplot as plt
```

In [4]:
```
tr = pd.read_table('.//drugsComTrain_raw.tsv')
```

In [5]:
```
tr.head()
```

Out[5]:

	Unnamed: 0	drugName	condition	review	rating	date	usefulCou
0	206461	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9.0	May 20, 2012	
1	95260	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8.0	April 27, 2010	1
2	92703	Lybrel	Birth Control	"I used to take another oral contraceptive, wh...	5.0	December 14, 2009	
3	138000	Ortho Evra	Birth Control	"This is my first time using any form of birth...	8.0	November 3, 2015	
				"Suboxone			

4	35696	Buprenorphine / naloxone	Opiate Dependence	has completely turned my life around...	9.0	November 27, 2016

There are five potential independent variables, one target variable, and a unique id column, for total of 7 columns.

In [6]:
```
tr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161297 entries, 0 to 161296
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   Unnamed: 0   161297 non-null  int64  
 1   drugName     161297 non-null  object 
 2   condition    160398 non-null  object 
 3   review       161297 non-null  object 
 4   rating       161297 non-null  float64
 5   date         161297 non-null  object 
 6   usefulCount  161297 non-null  int64  
dtypes: float64(1), int64(2), object(4)
memory usage: 8.6+ MB
```

There are very few missing values, only 'condition' has missing values of about 1,000, less than 1% of total sample.

In [7]:
```
trd = tr.dropna()
```

In [8]:
```
trd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 160398 entries, 0 to 161296
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   Unnamed: 0   160398 non-null  int64  
 1   drugName     160398 non-null  object 
 2   condition    160398 non-null  object 
 3   review       160398 non-null  object 
 4   rating       160398 non-null  float64
 5   date         160398 non-null  object 
 6   usefulCount  160398 non-null  int64  
dtypes: float64(1), int64(2), object(4)
memory usage: 9.8+ MB
```

Dealing with missing values.

Removing nonsensical samples in condition by creating new, clean column.

In [9]:
```
trd['condC']= trd['condition'].map(lambda x: x if "users" not in str(x) else
```

```
C:\Users\jmark\AppData\Local\Temp\ipykernel_24648\1641903905.py:1: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
```

```
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  trd['condC']= trd['condition'].map(lambda x: x if "users" not in str(x) else
'')
```

In [10]:
```
 trd= trd.drop('condition', axis = 1)
```

In [11]:
```
trd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 160398 entries, 0 to 161296
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   Unnamed: 0   160398 non-null  int64  
 1   drugName     160398 non-null  object 
 2   review       160398 non-null  object 
 3   rating       160398 non-null  float64
 4   date         160398 non-null  object 
 5   usefulCount  160398 non-null  int64  
 6   condC        160398 non-null  object 
dtypes: float64(1), int64(2), object(4)
memory usage: 9.8+ MB
```

It initially appears there are 812 unique condtions, but some entries may overlap (i.e. heart failure /attack and different types of diabetes)

In [12]:
```
len(trd.condC.unique())
```

Out[12]: 812

In [13]:
```
count = 0
sets = {}
for i in trd.condC:
    if sets.get(i, 0) >0:
        sets[i]+=1
    else:
        sets[i]=1
```

In [14]:
```
len(sets.values())
```

Out[14]: 812

The data is fairly balanced by drug, uniformly distributed.

In [15]:
```
count = 0
sets = {}
for i in trd.drugName:
    if sets.get(i, 0) >0:
        sets[i]+=1
    else:
        sets[i]=1
```

In [16]:
```
len(sets)
```

Out[16]: 3431

In [17]:
```
len(set(trd.drugName.sort values()))
```

Out[17]: 3431

It initially appears there are 3431 unique drugs, however names may overlap with eachother, and not be so distinct.

Written review metrics:

In [18]:
```
sm=0
for i in range(10):
    sm += len(trd['review'][i])


sm/11/5
```

Out[18]: 90.0

About 90 words per review, given sample of 11 and average 5 words per sentence.

Example below:

In [19]:
```
trd.review[22]
```

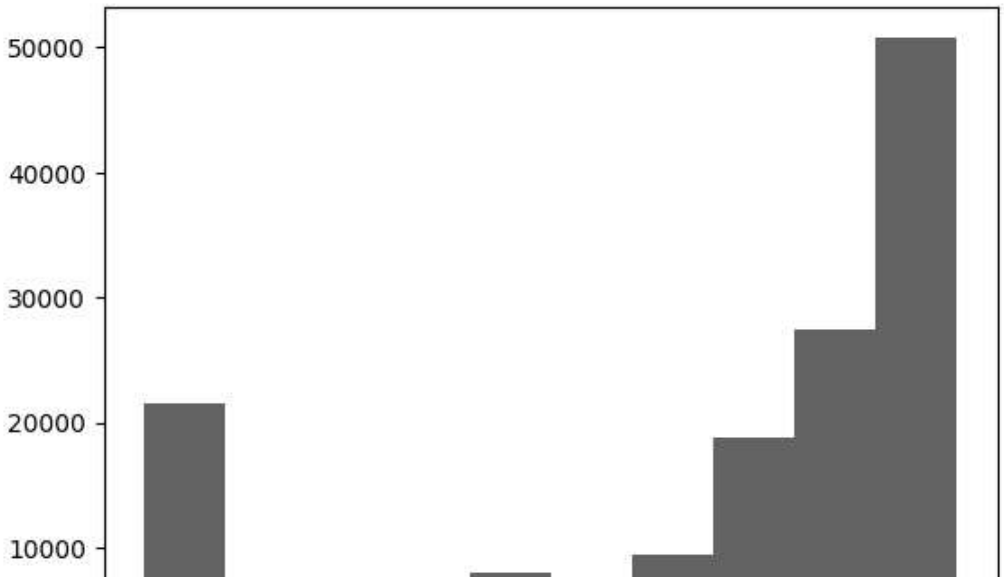
Out[19]:
```
'"Nexplanon does its job. I can have worry free sex. The only thing is that m
y periods are sometimes light and sometimes heavy. Sometimes they go away and
sometimes they show up unexpected. I also feel somewhat depressed. Not sure i
f its Nexplanon or not. I&#039;ve had Nexplanont for about 2 months now, but
despite the side effects its the most effective birth control I&#039;ve ever
used and I do not plan on taking it out."'
```

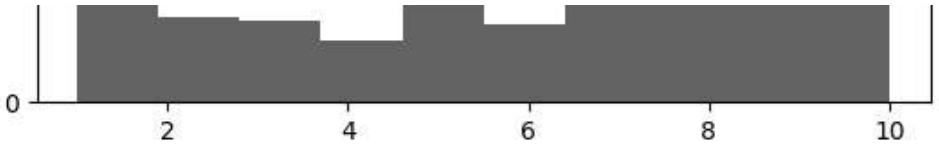
Below is a historgam of the patients' ratings.

In [20]:
```
fig, ax =plt.subplots()

ax.hist(trd['rating'])
```

Out[20]:
```
(array([21504.,  6879.,  6465.,  4980.,  7959.,  6301.,  9395., 18791.,
        27379., 50745.]),
 array([ 1. ,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1, 10. ]),
 <BarContainer object of 10 artists>)
```





Ratings are not normally distributed. Counts are highest at the worst and best ratings.

Data processing

Data processing involves tokenizing and then performing either tf-idf or word embedding processing on the tokens.

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [22]:
```
yrat = trd.rating
trdd = trd.drop('rating', axis = 1)
```

In [23]:
```
rev = trd.review
```

In [24]:
```
#from nltk import word_tokenize
from nltk.tokenize import RegexpTokenizer

basic_token_pattern = r"(?u)\b\w\w+\b"

tokenizer = RegexpTokenizer(basic_token_pattern)
```

In [ ]:
```

```

In [25]:
```
from nltk import FreqDist
```

First tokenizing, then creating the distribution frequencies for all invidual items.

In [26]:
```
def rowFrs(reviews):
    Frds= []

    for i in trd.review:
        lw= i.lower()
        tkn = tokenizer.tokenize(lw)
        Frd= FreqDist(tkn)

        Frds.append(Frd)
    return Frds
```

In [27]:
```
Frqs= rowFrs(rev)
```

In [28]:
```
len(Frqs)
```

Out[28]:
```
160398
```

Expand below to include all rows, i.e. creating the frequency distribution for the corpus.

Also choose a sub-sample space for this project of 16,000 (10% of total) for better computing.

In [29]:
```
com = trd[0:16000]
```

In [30]:
```
comb = ""
for i in com['review']:
    comb += str(i)
```

In [31]:
```
comb = comb.lower()
```

In [32]:
```
ttestw= tokenizer.tokenize(comb)
```

In [33]:
```
from nltk.corpus import stopwords
```

In [34]:
```
stopwords_list = stopwords.words('english')

w_words_stopped = [word for word in ttestw if word not in stopwords_list]
```

In [35]:
```
w_words_stoppedC = [word for word in w_words_stopped if '039' not in word]
```

In [36]:
```
FD= FreqDist(w_words_stoppedC)
```

Converting to dataframe to sort and get top 200 words.

In [37]:
```
df = pd.DataFrame(data = dict(FD), index = [0])#range16k
```

In [38]:
```
dft=df.transpose()
```

In [39]:
```
dft.head()
```

Out[39]:

	0
side	5259
effect	1066
take	4217
combination	145
bystolic	22

In [40]:
```
dft.sort_values(by= 0, axis = 0, ascending = False)
```

Out[40]:

	0
day	5810
side	5259

taking	5085
years	4649
pain	4519
...	...
lobular	1
peritoneum	1
acknowledged	1
ai	1
inflammed	1

20209 rows × 1 columns

In [41]:
```
df200 =dft.sort_values(by= 0, axis = 0).tail(200)
```

Adic is the inverse frequency part of tf-idf.

In [42]:
```
adic={}
for i in range(len(df200[0])):
    indx =df200.index[i]
    adic[indx]=df200[0][i]
```

Data modeling

Baseline tf-idf model.

In [43]:
```
from sklearn.metrics import mean_squared_error
```

In [44]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [45]:
```
from sklearn.ensemble import RandomForestRegressor
```

Creating the tf-idf matrix.

In [46]:
```
wtfids = []
for i in range(16000):
    tfidfs =list([])
    for k in adic.keys():
        num = Frqs[i].get(k,0)
        den = adic[k]
        tfidf = num/den
        tfidfs.append(tfidf)
    wtfids.append(tfidfs)
```

Taking the first 16000 of the y-var (rating).

In [47]:
```
revscut= trd.rating[0:16000]
```

In [48]:
```
yvals = np.asarray(revscut)
```

In [49]:
```
xvals = np.asarray(wtfids)
```

In [50]:
```
x_train, x_val, y_train,  y_val = train_test_split(xvals, yvals, random_state
```

Try baseline regression tree.

In [51]:
```
regrtr1 = DecisionTreeRegressor(max_depth = 3)
```

In [52]:
```
regrtr1.fit(x_train, y_train)
```

Out[52]:
```
DecisionTreeRegressor(max_depth=3)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:
```
prds4 = regrtr1.predict(x_train)
```

In [54]:
```
mean_squared_error(y_train, prds4)
```

Out[54]:
```
10.176464790716771
```

In [55]:
```
rmse = mean_squared_error(y_train, prds4)**.5
rmse
```

Out[55]:
```
3.190057176715924
```

In [56]:
```
regrtr1.feature_importances_
```

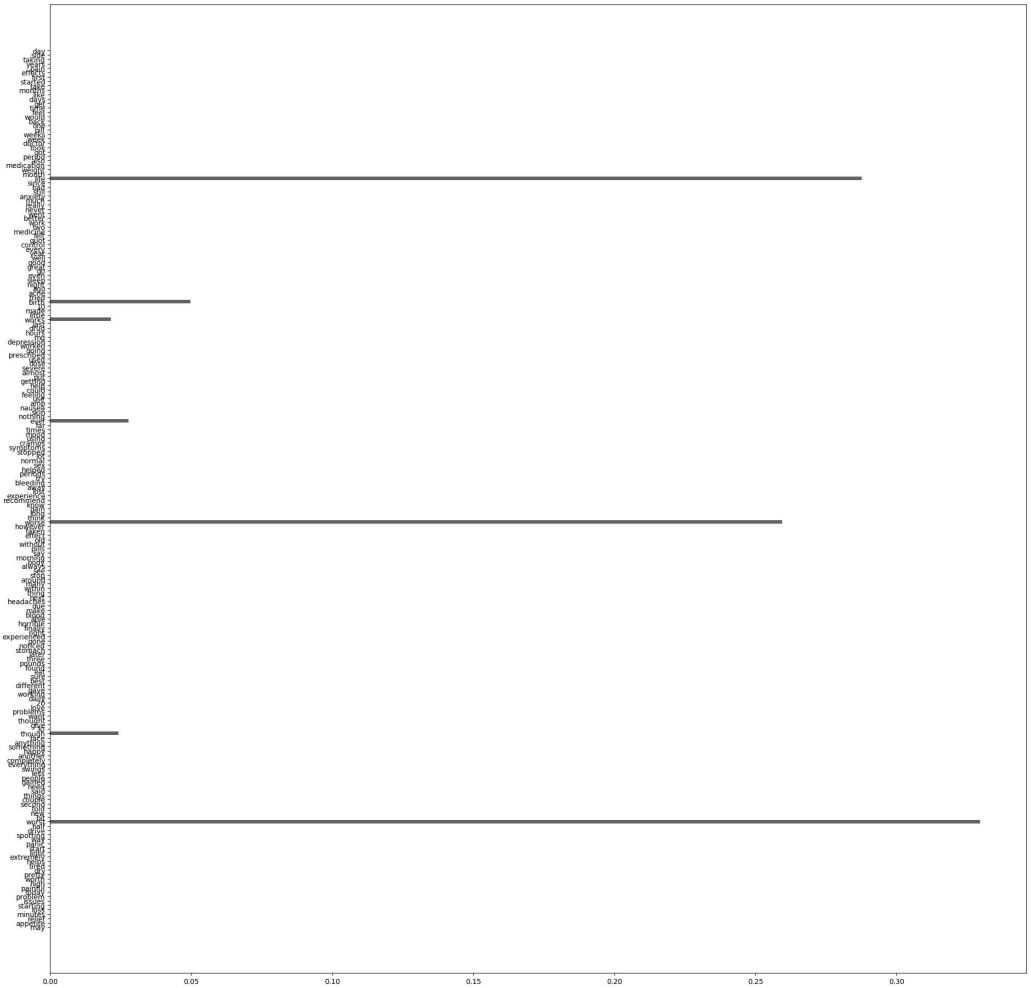
Out[56]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.32960934,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.02415096,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.25952593, 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.02778917, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02157691, 0.        ,
0.        , 0.        , 0.04972054, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.28762715, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ])
```

In [57]:

```
#fig, ax =fig
plt.figure(figsize=(25,25))
plt.barh(list(adic.keys()), regrtr1.feature_importances_)
```

Out[57]: <BarContainer object of 200 artists>



Most important features: worse, love

In [ ]:

In [58]:
```
prds4val = regrtr1.predict(x_val)
```

Baseline mse is 8, scale is 10 points.

In [59]:
```
mean_squared_error(y_val, prds4val)
```

Out[59]: 9.898097270874588

In [60]:
```
rmse = mean_squared_error(y_val, prds4val)**.5
rmse
```

Out[60]: 3.1461241664744555

Again, not much overfitting. Try tree with greater depth to reduce underfitting:

In [61]:
```
regrtr1a = DecisionTreeRegressor(max_depth = 5)
```

In [62]:
```
regrtr1a.fit(x_train, y_train)
```

Out[62]: DecisionTreeRegressor(max_depth=5)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [63]:
```
prds4 = regrtr1a.predict(x_train)
```

In [64]:
```
mean_squared_error(y_train, prds4)
```

Out[64]: 9.693739045289481

In [65]:
```
rmse = mean_squared_error(y_train, prds4)**.5
rmse
```

Out[65]: 3.1134770025310097

In [66]:
```
prds4val = regrtr1a.predict(x_val)
```

In [67]:
```
mean_squared_error(y_val, prds4val)
```

Out[67]: 9.586180223941854

In [68]:
```
rmse = mean_squared_error(y_val, prds4val)**.5
rmse
```

Out[68]: 3.096155717004856

Allowing for additional depth reduced underfitting slightly

Allowing for additional depth, reduced underfitting, slightly.

Running a linear regression model:

In [69]:
```
from sklearn import linear_model
lreg = linear_model.LinearRegression()
```

In [70]:
```
lreg.fit(x_train, y_train)
```

Out[70]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [71]:
```
preds = lreg.predict(x_train)
```

In [72]:
```
mean_squared_error(y_train, preds)
```

Out[72]:
```
8.141032346511635
```

In [73]:
```
rmse = mean_squared_error(y_train, preds)**.5
rmse
```

Out[73]:
```
2.853249436434122
```

In [74]:
```
valpreds = lreg.predict(x_val)
```

In [75]:
```
mean_squared_error(y_val, valpreds)
```

Out[75]:
```
8.12230730226126
```

In [76]:
```
rmse = mean_squared_error(y_val, valpreds)**.5
rmse
```

Out[76]:
```
2.8499661931786595
```

Linear regression is more accurate than the baseline tree regressions. Not a lot, if any, overfitting, likely not underfitting either (2.84 rmse training and validation)

In [77]:
```
rfr = RandomForestRegressor(max_samples = 100)
```

In [78]:
```
rfr.fit(x_train, y_train)
```

Out[78]:
```
RandomForestRegressor(max_samples=100)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [79]:
```
prds5 = rfr.predict(x_train)
```

In [80]:
```
mean_squared_error(y_train, prds5)
```

Out[80]: 10.071233960416665

In [81]:
```
rmse = mean_squared_error(y_train, prds5)**.5
rmse
```

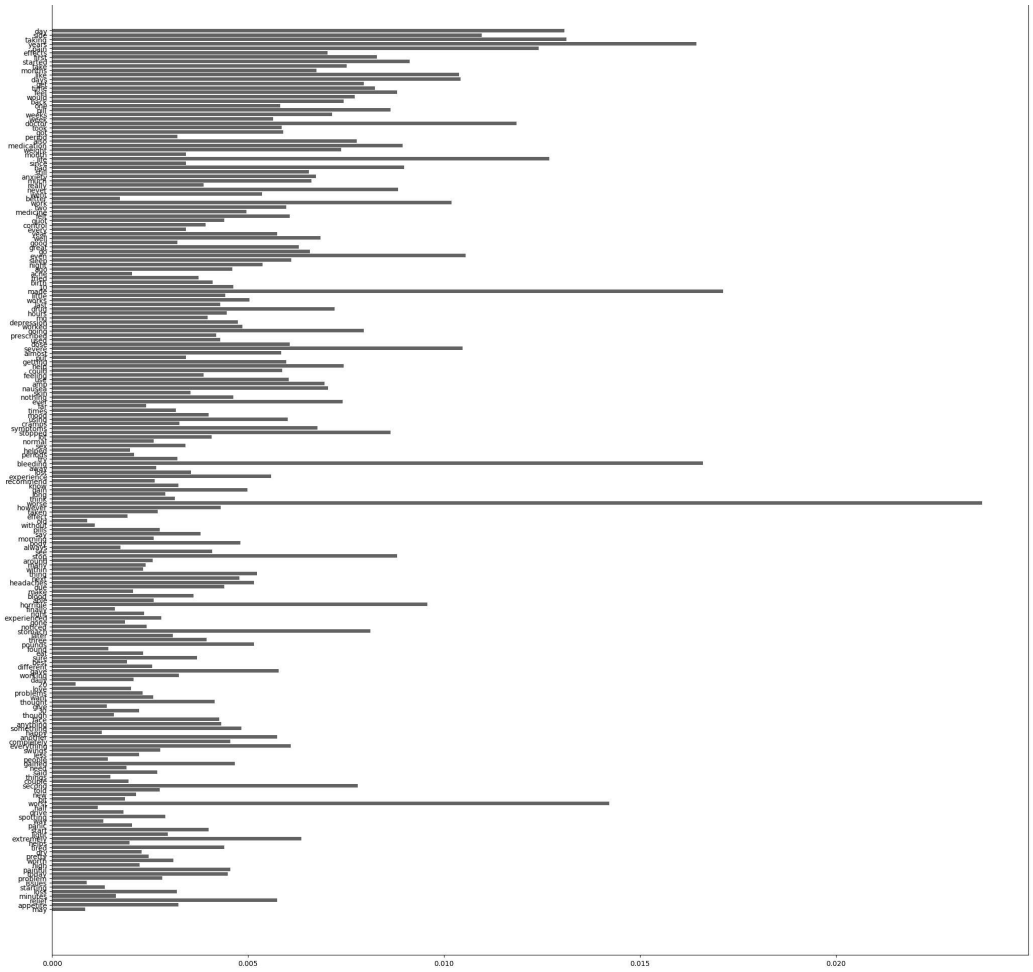
Out[81]: 3.173520751533959

In [82]:
```
rfr.feature_importances_
```

Out[82]:
```
array([0.0008406 , 0.00322102, 0.00574013, 0.00162874, 0.00317727,
       0.00134474, 0.00087997, 0.00280677, 0.00448227, 0.00454533,
       0.00223067, 0.00309843, 0.00246063, 0.00228296, 0.00439586,
       0.00197706, 0.0063614 , 0.00295721, 0.00399943, 0.00203815,
       0.00130691, 0.00288135, 0.00181689, 0.001159  , 0.01422138,
       0.00186075, 0.00214756, 0.00275111, 0.00780158, 0.00194659,
       0.00148729, 0.00268016, 0.00190328, 0.00466585, 0.00142639,
       0.00221583, 0.00275438, 0.00609079, 0.00454232, 0.00574457,
       0.00126725, 0.00483349, 0.00430937, 0.00426494, 0.00158026,
       0.00221705, 0.00139153, 0.00415194, 0.00258296, 0.00230289,
       0.002017  , 0.00059623, 0.00207771, 0.00323387, 0.00578399,
       0.00255829, 0.00190542, 0.00369207, 0.00231584, 0.00143125,
       0.00514583, 0.00394226, 0.00307804, 0.00811737, 0.00241455,
       0.00186072, 0.00278494, 0.00234448, 0.00160365, 0.00957234,
       0.00259778, 0.00361223, 0.00207032, 0.00438875, 0.00515036,
       0.00477992, 0.00522242, 0.00231821, 0.0023834 , 0.00256428,
       0.00880216, 0.00408111, 0.00173808, 0.00479698, 0.00259692,
       0.00379374, 0.0027454 , 0.00108978, 0.0008883 , 0.00192614,
       0.00269467, 0.00430203, 0.02372881, 0.00313027, 0.00288503,
       0.00498266, 0.0032185 , 0.00262365, 0.00558726, 0.00354241,
       0.00265592, 0.01661123, 0.00319669, 0.00209548, 0.0019848 ,
       0.00340113, 0.00259033, 0.00406446, 0.00863249, 0.00677377,
       0.00324389, 0.00600818, 0.00399689, 0.00315621, 0.00239442,
       0.00741204, 0.00461923, 0.00353211, 0.00704619, 0.00695656,
       0.00603743, 0.00385938, 0.0058719 , 0.00744319, 0.00597361,
       0.00341001, 0.00584661, 0.01047569, 0.00605784, 0.00428304,
       0.00418999, 0.00795084, 0.00486118, 0.00474336, 0.00396331,
       0.00445077, 0.00720347, 0.00429031, 0.00502901, 0.00441755,
       0.01711662, 0.00462457, 0.00409989, 0.00373848, 0.00204048,
       0.00459802, 0.00536862, 0.00610358, 0.01055287, 0.00657269,
       0.00629969, 0.00319998, 0.00684704, 0.00574832, 0.00341711,
       0.00391788, 0.00439344, 0.00605827, 0.00495446, 0.00597961,
       0.01018568, 0.00172397, 0.00535767, 0.00882777, 0.00387036,
       0.00661656, 0.00672751, 0.00654853, 0.00898509, 0.00340823,
       0.01268883, 0.0034083 , 0.00736915, 0.00893999, 0.00777957,
       0.00319013, 0.0058922 , 0.00586372, 0.01185605, 0.00564122,
       0.00714318, 0.00863445, 0.00581618, 0.00743477, 0.0077171 ,
       0.00880436, 0.00823851, 0.00795711, 0.01042478, 0.01038231,
       0.00675028, 0.0075225 , 0.00912462, 0.00828832, 0.00702767,
       0.01241305, 0.01643607, 0.01312608, 0.01095753, 0.01306676])
```

In [83]:
```
plt.figure(figsize=(25,25))
plt.barh(list(adic.keys()), rfr.feature_importances_)
```

Out[83]: <BarContainer object of 200 artists>



Important features: horrible, worst, doctor

In [84]:
```
prds5val = rfr.predict(x_val)
```

In [85]:
```
mean_squared_error(y_val, prds5val)
```

Out[85]: 9.895772574999999

In [86]:
```
rmse = mean_squared_error(y_val, prds5val)**.5
rmse
```

Out[86]: 3.14575469084924

Linear regression continues to perform better.

word embeddings, premade

Use word-embeddings to process input data so that some word-meaning is caputured. Use Stanford's premade word-embedding, "Glove". The W2vectorizer object has method, transorm, which creates an array that has the glove vector for words in the glove dictionary, and rows of 0's for words not in the glove dictionary.

In [87]:
```
total_vocabulary = set(word for word in w_words_stoppedC)
```

In [88]:
```
len(total_vocabulary)
```

Out[88]: 20209

In [89]:
```
glove = {}
with open('glove.6B.50d.txt', 'rb') as f:
    for line in f:
        parts = line.split()
        word = parts[0].decode('utf-8')
        if word in total_vocabulary:
            vector = np.array(parts[1:], dtype=np.float32)
            glove[word] = vector
```

In [90]:
```
class W2vVectorizer(object):

    def __init__(self, w2v):
        # Takes in a dictionary of words and vectors as input
        self.w2v = w2v
        if len(w2v) == 0:
            self.dimensions = 0
        else:
            self.dimensions = len(w2v[next(iter(glove))])

    def fit(self, X, y):
        return self
    # Gets the mean vector from all different words in the particular review.
    def transform(self, X):
        return np.array([
            np.mean([self.w2v[w] for w in words if w in self.w2v]
                    or [np.zeros(self.dimensions)], axis=0) for words in X])
```

In [91]:
```
t1 = W2vVectorizer(glove)
```

In [92]:
```
t1.dimensions
```

Out[92]: 50

In [93]:
```
t2 = t1.transform(Frqs)
```

In [94]:
```
np.shape(t2)
```

Out[94]: (160398, 50)

In [95]:
```
ys=trd.rating[0:16000]
t2 =t2[0:16000]
```

In [96]:
```
x_trainW, x_valW, y_trainW,  y_valW = train_test_split(t2, ys, random_state =
```

In [ ]:
```

```

In [97]:
```
reglW = linear_model.LinearRegression()
```

In [98]:
```
reglW.fit(x_trainW, y_trainW)
```

Out[98]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [99]:
```
predsW = reglW.predict(x_trainW)
```

In [100…
```
mean_squared_error(y_trainW, predsW)
```

Out[100…
```
9.3175497555418
```

In [101…
```
rmse = mean_squared_error(y_trainW, predsW)**.5
rmse
```

Out[101…
```
3.0524661759865253
```

In [102…
```
predsW = reglW.predict(x_valW)
```

In [103…
```
mean_squared_error(y_valW, predsW)
```

Out[103…
```
9.242511595713813
```

In [104…
```
rmse = mean_squared_error(y_valW, predsW)**.5
rmse
```

Out[104…
```
3.040149929808366
```

Linear regression model without embeddings still performing better.

In [105…
```
from sklearn.tree import DecisionTreeRegressor
```

In [106…
```
regrtrw = DecisionTreeRegressor(max_depth = 3)
```

In [107…
```
regrtrw.fit(x_trainW, y_trainW)
```

Out[107…
```
DecisionTreeRegressor(max_depth=3)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [108…
```
prds3w = regrtrw.predict(x_trainW)
```

In [109…
```
mean_squared_error(y_trainW, prds3w)
```

Out[109… 10.059328144618801

In [110…
```
rmse = mean_squared_error(y_trainW, prds3w)**.5
rmse
```

Out[110… 3.171644391261227

In [111…
```
prdsvalW = regrtrw.predict(x_valW)
```

In [112…
```
mean_squared_error(y_valW, prdsvalW)
```

Out[112… 9.963422304715259

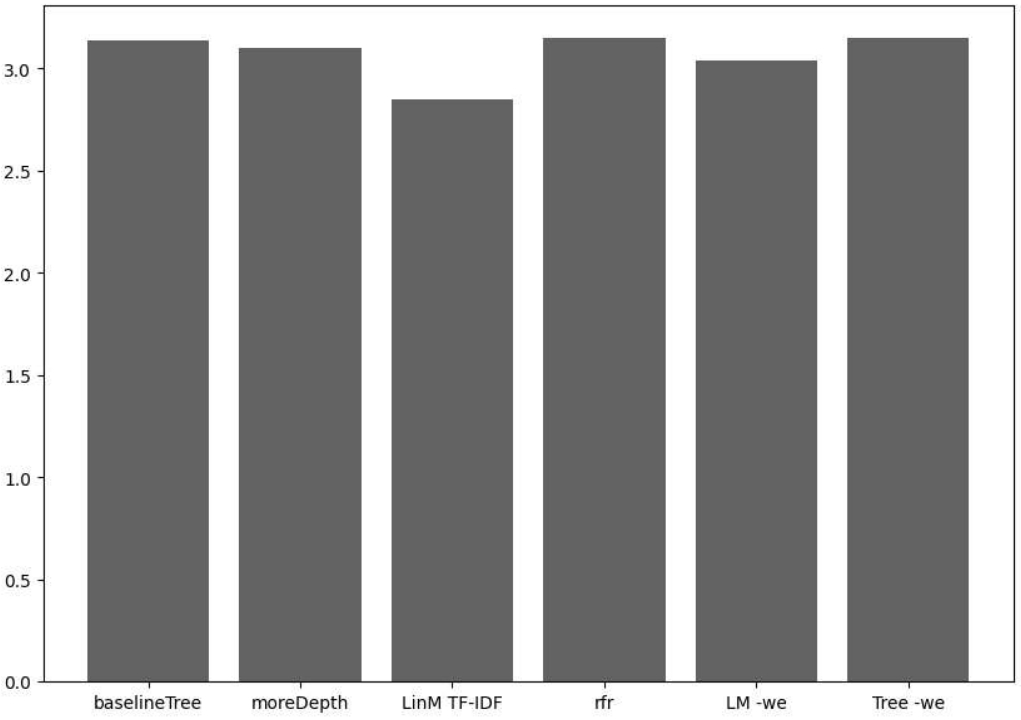
In [113…
```
rmse = mean_squared_error(y_valW, prdsvalW)**.5
rmse
```

Out[113… 3.156488920416997

Linear regression model still performing better. the linear regression with (TF-IDF) is the only model with an rmse less than 3.0.

In [139…
```
fig, ax = plt.subplots(figsize = (10,7))
ax.bar(x= ['baselineTree', 'moreDepth','LinM TF-IDF', 'rfr', 'LM -we','Tree -
```

Out[139… <BarContainer object of 6 artists>



Evaluate the linear regression model on unseen data.

In [114…
```
# getting fd for second 16000 samples of data.
FrqsE = Frqs[16000:32000]
```

In [115…

```
wtfids = []
for i in range(16000):
    tfidfs =list([])
    for k in adic.keys():
        num = FrqsE[i].get(k,0)
        den = adic[k]
        tfidf = num/den
        tfidfs.append(tfidf)
    wtfids.append(tfidfs)
```

Taking the second 16000 of the y-var (rating).

In [116…

```
revscut= trd.rating[16000:32000]
```

In [117…

```
yvals = np.asarray(revscut)
```

In [118…

```
xvals = np.asarray(wtfids)
```

In [119…

```
Epreds = lreg.predict(wtfids)
```

In [120…

```
mean_squared_error(yvals, Epreds)
```

Out[120… 8.30875077189119

In [121…

```
mean_squared_error(yvals, Epreds)**.5
```

Out[121… 2.88249037672135

Results/conclusions

rmses

- Deploy linear regression (TF-IDF) for 'rating extraction' from written review.
- Gather insights on how patients rate drugs.
  - "doctor, love, worse" etc.
- Combine the tf-idf and word embedding models.
- Use the "meta-data" as features. (i.e. the drug evaluated)

Repository structure

- Notebook
- README
- Presentation