Project Report DSE 220

Kaggle Name : Akash Shah

Problem statement:

The problem statement is to predict the number of votes that are helpful out of the total number of votes given to an amazon product review. The problem looks like a classification problem but is actually a bounded regression model. All the following steps taken in order will help us get themae score of 0.16452 on the public leaderboard. While the sequence of steps seems to be linear below, it has come after rounds of experimentation, not necessarily in this order.

Pre-Processing:

Pre-processing has taken a major chunk of the time required for this problem. The dimensionality of the problem seems low at the start with just around 12 features. Eventually we realize that it's not the case and the following steps help us get much better and meaningful features.

1. Item Encoding: Item IDs are present as strings like 'I655355328' . They do not contain too much information as strings alone. But it would be unfair to ignore them as some items might be more expensive than others, and people might find reviews more helpful in that case. Hence using the LabelEncoder library in sklearn we encode them to unique integers.
2. Reviewer Encoding: Reviewer IDs are present as strings just like ItemIDs but we might see reviewer IDs in the test data that are not seen in the training data. Hence we classify all the unseen reviewerIDs as 'unknown' and add this label to our label_encoder.
3. Date: There are two dates that turn out to be the same . UnixReviewTime is the epoch time and reviewTime is the string version of the same time. We split this data into day, month and year for each review, considering that each of these might be individually important.
4. TextLength: This feature uses the reviewText feature. We count the number of words of the review as the length. This seems to have a real impact on the model and reduces mae by quite a lot.
5. Categories: Initially dropping categories was tried , but it resulted in an increase of mae. Some sort of flattening of the categories was required. We have flattened the categories to obtain a one-hot encoding (not exactly)like representation using multiLabelBinarizer in sklearn.
6. Drop columns:
    a. Price: The price column consists of 63% of null values. This column proves to be inefficient when tried with a couple of models and did not add much information, even though we might think that it would. Thus after significant experimentation this column was removed.
    b. reviewTime: Dropped because we divided this into day,month and year.

c. unixReviewTime: Dropped because it was the same as reviewTime
d. ReviewHash: The ID of the review added no information and would be unique for each row.
e. Helpful: We split this column into 2: outOf and nHelpful (dependent variable) and drop it.
f. Summary: Many NLP processes like tf-idf and Countvectorizer were tried on this and rejected, as the dimensionality of the data exploded and the summary did not
g. reviewText: While there can be a lot done with this , we have not found tf-idf and Countvectorizer to be particularly useful for this. It has only caused the curse of dimensionality and the down performance of the model.

We have successfully converted our data into integers and floats . We tried using a standard scalar for normalizing our data, but it was not necessary as we ended up using a Bagging model in the end.

Data Splitting Strategy:

Data was split into train , validation and test. (Test was already given). We used 20% of the training data for validation, amounting to 160000 and 40000 examples respectively, in order to reduce overfitting on the test data. Only when we obtain a mean absolute error on validation do we go for test submission.

Model Selection:

Model selection has been pretty straightforward. The following models were tried and rejected with the reasons state:

1. Linear Regression: really high mae of > 0.3
2. Dense Fully Connected Neural Networks - Really poor convergence and difficult to interpret why. The network kept predicting the same value in order to minimize loss.
3. Recurrent Neural Network: An LSTM applied to the reviewText followed by a concatenation to the original features also was tried, but in interest of time could not be optimized.
4. SVM Regressor: while this could have fit, it took a very long time to train an SVM Regressor with gaussian kernel, which made it impossible to experiment and tune.
5. DecisionTreeRegressor: Too simplistic and resulted in overfitting. Really high mae values for our problem

We eventually found randomForestRegressor to be the best for our data, it could use the highly categorical features quite well, and using default parameters on train data we could reach a mae value of 0.17445 on validation data.

Final Model:

In order to select our final model we have performed a grid search. However not using the sklearn library. It has come down to trying various permutations and combinations of the parameters. We finally select -

RandomForestRegressor

Hyperparameters:

1. n_jobs = -1 . Helps us train faster and optimize as per machine
2. Max_depth = default (full trees). We have left max_depth to full trees in order to avoid the model from underfitting and generalizing too much.
3. N_estimators: By far the most important hyperparameter. We observe the best predictions at value 250 estimators, the more the estimators the better, but we also wanted to save on training time.

```
# Re-training with entire training data
rf = RandomForestRegressor(n_estimators = 250,verbose = 1,n_jobs = -1)
rf.fit(train,help_)
```

Predictions:

An important operation that has helped this application is the rounding off of predictions. There is no regressor that would return integers , and the number of votes would never be a decimal value. In order to obtain better mae values we have performed rounding of these values depending on whether the value is above or below 0.5

```
def smooth_predictions(preds):
    import numpy as np
    for i in range(0,preds.shape[0]):
        x = preds[i]
        if x % 1 > 0.5:
            preds[i] = np.ceil(x)
        else:
            preds[i] = np.floor(x)
    return preds
```

Conclusion:

We obtain a mean absolute error of 0.16452 on 60% of test data with this approach, and hope that it generalizes well to the rest as well.

final_new

June 14, 2020

[180]:
```
import gzip
import pandas as pd
from collections import defaultdict

def readGz(f):
    for l in gzip.open(f):
        yield eval(l)

def parse(path):
    g = gzip.open(path, 'rb')
    for l in g:
        yield eval(l)

def getDF(path):
    i = 0
    df = {}
    for d in parse(path):
        df[i] = d
        i += 1
    return pd.DataFrame.from_dict(df, orient='index')

train_df = getDF('train.json.gz')
test_df = getDF('test_Helpful.json.gz')
```

[181]:
```
train_df.head(5)
```

[181]:

	categoryID	categories	itemID	reviewerID	rating	reviewText
0	0	[[Clothing, Shoes & Jewelry, Women], [Clothing…	I655355328	U745881038	3.0	These are cute, but they are a little small. …
1	0	[[Clothing, Shoes & Jewelry, Women, Clothing, …	I241092314	U023577405	4.0	I love the look of this bra, it is what I want…
2	0	[[Clothing, Shoes & Jewelry, Wedding Party Gif…	I408260822	U441384838	3.0	it's better on a man's hand.I didn't find it v…
3	0	[[Clothing, Shoes & Jewelry, Women, Clothing, …	I770448753			
4	0	[[Clothing, Shoes & Jewelry, Women, Plus-Size,…	I919238161			

```
3  U654041297     4.0  Comfortable and easy to wear for a day of shop…
4  U096604734     5.0  I'm quite small and the XS fits me like a regu…

   reviewHash   reviewTime                        summary  unixReviewTime  \
0  R115160670  05 20, 2014                           Cute      1400544000
1  R800651687   02 7, 2013  Beautiful but size runs small      1360195200
2  R345042616  05 13, 2014              Good price but…      1399939200
3  R875466866  05 25, 2014                   Easy, breezy      1400976000
4  R317526520  07 30, 2013                    Great shirt      1375142400

                      helpful  price
0  {'outOf': 0, 'nHelpful': 0}    NaN
1  {'outOf': 0, 'nHelpful': 0}    NaN
2  {'outOf': 2, 'nHelpful': 2}  19.99
3  {'outOf': 0, 'nHelpful': 0}  14.95
4  {'outOf': 1, 'nHelpful': 1}    NaN
```

[182]:
```
import itertools

train_df['categories'] = train_df.categories.apply(lambda x: list(itertools.
 ↪chain(*x))).apply(lambda x: [each.split(',') for each in x]).apply(lambda x:␣
 ↪list(set(itertools.chain(*x))))
test_df['categories'] = test_df.categories.apply(lambda x: list(itertools.
 ↪chain(*x))).apply(lambda x: [each.split(',') for each in x]).apply(lambda x:␣
 ↪list(set(itertools.chain(*x))))
```

[183]:
```
categories_fit = train_df.categories.to_list()
test_categories_fit = test_df.categories.to_list()
```

[184]:
```
from sklearn import preprocessing
lb = preprocessing.MultiLabelBinarizer()

lb.fit(categories_fit)
train_cats = pd.DataFrame(lb.transform(categories_fit),columns = lb.classes_)
test_cats = pd.DataFrame(lb.transform(test_categories_fit),columns = lb.
 ↪classes_)
```

[185]:
```
train_df.drop('categories',axis = 1,inplace = True)
test_df.drop('categories',axis = 1, inplace = True)
```

[186]:
```
train = pd.concat([train_df,train_cats],axis = 1)
test = pd.concat([test_df,test_cats],axis = 1)

# Uncomment to run without categories
#train = train_df
#test = test_df
```

[187]:
```
helpfulNess_train = pd.DataFrame(train_df.helpful.apply(lambda x:␣
↪[x['outOf'],x['nHelpful']]).to_list(),columns = ['outOf','nHelpful'])

helpfulNess_test = pd.DataFrame(test_df.helpful.apply(lambda x: [x['outOf']]).
↪to_list(),columns = ['outOf'])
```

[188]:
```
train = pd.concat([train,helpfulNess_train],axis = 1)
test = pd.concat([test,helpfulNess_test],axis = 1)
```

[189]:
```
train.drop(['helpful','reviewHash','reviewText','price'],axis = 1,inplace =␣
↪True)
test.drop(['helpful','reviewHash','reviewText','price'],axis = 1,inplace = True)
```

[190]:
```
month = train.reviewTime.apply(lambda x: [each.replace(',','') for each in␣
↪str(x).split(' ')]).apply(lambda x:x[0])
day = train.reviewTime.apply(lambda x: [each.replace(',','') for each in str(x).
↪split(' ')]).apply(lambda x:x[1])
year = train.reviewTime.apply(lambda x: [each.replace(',','') for each in␣
↪str(x).split(' ')]).apply(lambda x:x[2])

month_test = test.reviewTime.apply(lambda x: [each.replace(',','') for each in␣
↪str(x).split(' ')]).apply(lambda x:x[0])
day_test = test.reviewTime.apply(lambda x: [each.replace(',','') for each in␣
↪str(x).split(' ')]).apply(lambda x:x[1])
year_test = test.reviewTime.apply(lambda x: [each.replace(',','') for each in␣
↪str(x).split(' ')]).apply(lambda x:x[2])
```

[191]:
```
train.drop(['reviewTime'],axis = 1,inplace = True)
test.drop(['reviewTime'],axis = 1,inplace = True)
train.drop(['unixReviewTime'],axis = 1,inplace = True)
test.drop(['unixReviewTime'],axis = 1,inplace = True)

train['day'] = day.astype('int')
train['month'] = month.astype('int')
train['year'] = year.astype('int')

test['day'] = day_test.astype('int')
test['month'] = month_test.astype('int')
test['year'] = year_test.astype('int')
```

0.0.1 Adding a feature for how many words there are in a Review

[192]:
```
import nltk
nltk.download('punkt')

train['textLength'] = train_df.reviewText.apply(lambda x:len(nltk.
 ↪word_tokenize(x)))
test['textLength'] = test_df.reviewText.apply(lambda x:len(nltk.
 ↪word_tokenize(x)))

## Summary length does not work as well

#train['summaryLength'] = train_df.summary.apply(lambda x:len(x.split(' ')))
#test['summaryLength'] = test_df.summary.apply(lambda x:len(x.split(' ')))
```

```
[nltk_data] Downloading package punkt to /home/akash59/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[193]:
```
train.drop(['summary'],axis = 1,inplace = True)
test.drop(['summary'], axis = 1,inplace = True)
```

[194]:
```
#train['noSent'] = train_df.reviewText.apply(lambda x:len(nltk.
 ↪sent_tokenize(x)))
#test['noSent'] = test_df.reviewText.apply(lambda x:len(nltk.sent_tokenize(x)))
```

0.1 Items will never be new, but reviewers will be.

[195]:
```
from sklearn.preprocessing import LabelEncoder
l_item = LabelEncoder()
l_item.fit(train.itemID)
```

[195]:
```
LabelEncoder()
```

[196]:
```
train['itemID'] = l_item.transform(train.itemID)
```

[197]:
```
test['itemID'] = l_item.transform(test.itemID)
```

[198]:
```
l_review = LabelEncoder()
l_review.fit(train.reviewerID)

train['reviewerID'] = l_review.transform(train.reviewerID)
#test['reviewerID'] = l_review.transform(test.reviewerID)
```

[199]:
```
import numpy as np
l_review.classes_ = np.append(l_review.classes_,'Unknown')
```

[200]:
```
unknowns = set(test.reviewerID).difference(l_review.classes_)
test['reviewerID'] = test.reviewerID.apply(lambda x: 'Unknown' if x in unknowns␣
 ↪else x)
test['reviewerID'] = l_review.transform(test.reviewerID)
```

0.1.1 Count Vectorizer for summary column

[201]:
```
#from nltk.corpus import stopwords
```

[202]:
```
#from sklearn.feature_extraction.text import TfidfVectorizer

#nltk.download('stopwords')

#count_vectorizer = TfidfVectorizer(stop_words=stopwords.
 ↪words('english'))#,lowercase=True,ngram_range = (3,4),max_features =␣
 ↪50000,max_df = 0.5)
#train_text = count_vectorizer.fit_transform(train_df.reviewText)
#test_text = count_vectorizer.transform(test_df.reviewText)
```

0.1.2 No need to scale Random Forests

[203]:
```
# from sklearn.preprocessing import StandardScaler

# scalers = {}
# for i in range(train.shape[1]):
#     colname = train.columns[i]is
#     scalers[colname] = StandardScaler()
#     train[colname] = scalers[colname].fit_transform(train[colname].to_numpy().
 ↪reshape(-1,1))

# for i in range(test.shape[1]):
#     colname = test.columns[i]
#     test[colname] = scalers[colname].transform(test[colname].to_numpy().
 ↪reshape(-1,1))
```

[204]:
```
help_ = train['nHelpful']
train = train.drop('nHelpful',axis = 1)
```

[205]:
```
import scipy
from scipy.sparse import hstack

train = scipy.sparse.csr_matrix(train.values)
test = scipy.sparse.csr_matrix(test.values)
```

```
# Comment to remove summary
# train = hstack((train,train_text))
# test = hstack((test,test_text))
```

[206]:
```
from sklearn.model_selection import train_test_split

train_csr_new,val_new, train_y , val_y = train_test_split(train,help_,test_size␣
↪= 0.2,random_state = 42)
```

[207]:
```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(verbose = 1,n_jobs = -1)
rf.fit(train_csr_new,train_y)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 6 concurrent workers.
[Parallel(n_jobs=-1)]: Done  38 tasks      | elapsed:   58.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.4min finished
```

[207]:
```
RandomForestRegressor(n_jobs=-1, verbose=1)
```

[208]:
```
def smooth_predictions(preds):
    import numpy as np
    for i in range(0,preds.shape[0]):
        x = preds[i]
        if x % 1 > 0.5:
            preds[i] = np.ceil(x)
        else:
            preds[i] = np.floor(x)
    return preds
```

[209]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(val_y,smooth_predictions(rf.predict(val_new)))
```

```
[Parallel(n_jobs=6)]: Using backend ThreadingBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  38 tasks      | elapsed:    0.0s
[Parallel(n_jobs=6)]: Done 100 out of 100 | elapsed:    0.1s finished
```

[209]:
```
0.17445
```

[210]:
```
def make_predictions(preds,message):
    outof = pd.DataFrame(test_df.helpful.apply(lambda x: [x['outOf']]).
↪to_list(),columns = ['outOf'])['outOf']
    predictions = pd.DataFrame([test_df.reviewerID,test_df.itemID,outof,preds]).
↪T
    predictions.rename(columns = {'reviewerID':0,'itemID':1,'Unnamed 0':
↪2,'Unnamed 1':3},inplace = True)
```

```
    predictions[0] = predictions[0] + '-' + predictions[1] + '-'
    predictions[0] = predictions[0].apply(lambda x: x.strip())
    predictions[0] = predictions[0] + predictions['outOf'].apply(lambda x:␣
↪str(int(x)).strip())
    predictions.drop([1,'outOf'],axis = 1 ,inplace = True)
    predictions.rename(columns = {0:'userID-itemID-outOf',2:
↪'prediction'},inplace = True)
    predictions.to_csv('predictions_latest.csv',sep= ',',index = False)
    #!kaggle competitions submit -c dse220 -f predictions_first.csv -m␣
↪'Simplest Random Forsts ,without price ,category and anything'
```

[211]:
```
# Re-training with entire training data
rf = RandomForestRegressor(n_estimators = 250,verbose = 1,n_jobs = -1)
rf.fit(train,help_)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 6 concurrent workers.
[Parallel(n_jobs=-1)]: Done  38 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 188 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done 250 out of 250 | elapsed:  8.5min finished
```

[211]:
```
RandomForestRegressor(n_estimators=250, n_jobs=-1, verbose=1)
```

[212]:
```
preds = smooth_predictions(rf.predict(test))
```

```
[Parallel(n_jobs=6)]: Using backend ThreadingBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  38 tasks      | elapsed:    0.0s
[Parallel(n_jobs=6)]: Done 188 tasks      | elapsed:    0.1s
[Parallel(n_jobs=6)]: Done 250 out of 250 | elapsed:    0.1s finished
```

[213]:
```
make_predictions(preds,'nothing')
```

[214]:
```
### 0.16452
```

[ ]:
```

```