Machine Learning Approach to Product pricing

- Sooraj Mangalath Subrahmannian

1. Problem Statement

Mercari, Japan’s biggest community-powered shopping app has various online products and is struggling to do product pricing at scale. They want to do some price suggestions to sellers

Sweater A:

“Vince Long-Sleeve Turtleneck Pullover Sweater, Black, Women's, size L, great condition.”

Sweater B:

“St. John's Bay Long-Sleeve Turtleneck Pullover Sweater, size L, great condition”

- Clothing has strong seasonal pricing trends and is heavily influenced by brand names
- Electronics have fluctuating prices based on product specs.

How to price various products based on their description, brand name, category name, and item condition?

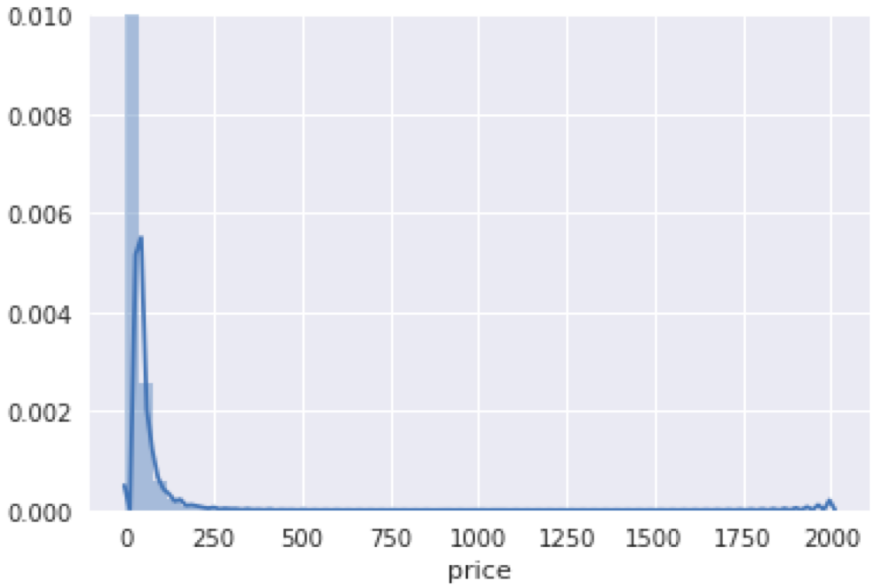
2. Data description

name - the title of the listing.
item_condition_id - the condition of the items provided by the seller
category_name - category of the listing
brand_name – name of the brand
price - the price that the item was sold for
item_description - the full description of the item.

Categorical Features	Number of unique values	No of records with null values
brand_name	4809	632682
category_name	1287	6327
item_condition_id	5	0
item description	1482535	4

Target Distribution	
Mean	26
Median	17
Min	0
Max	2009

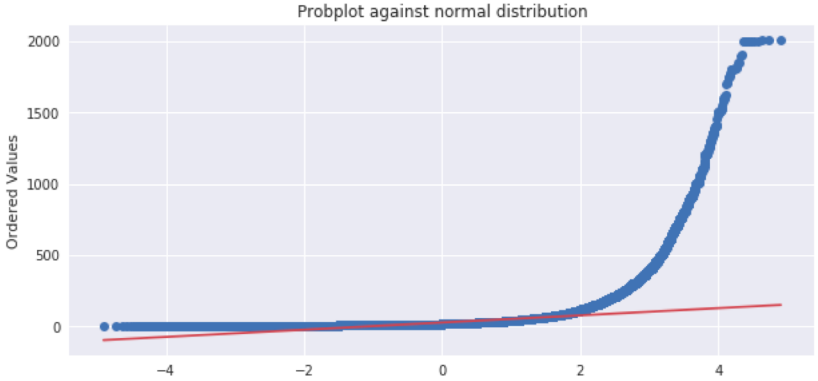
3.1 Target Distribution

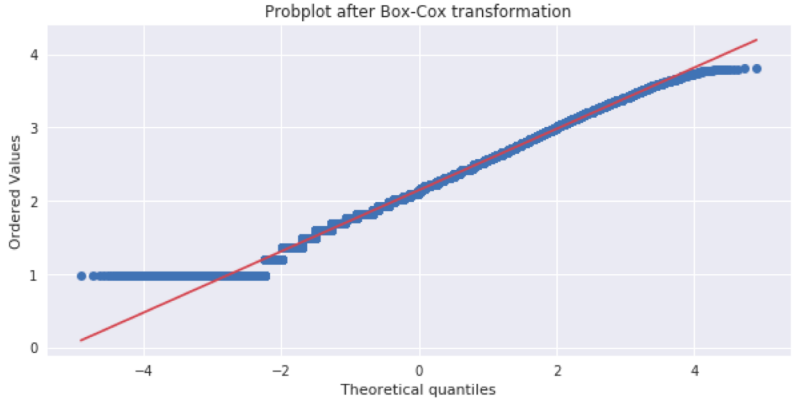


Issues:

- There were 870 records with price as zero
- The target distribution is heavily right skewed
- There were many outliers (extreme values in y)

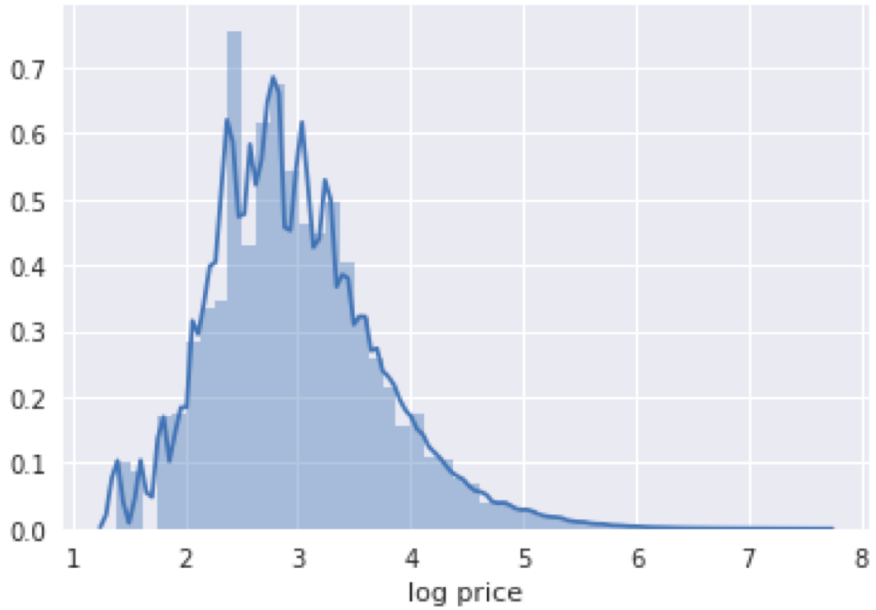
3.2 Target Distribution : Box-Cox Transformation






```
fig = plt.figure(figsize=(10,10))
ax1 = fig.add_subplot(211)
prob = st.probplot(train_df.price.values, dist=st.norm, plot=ax1)
ax1.set_xlabel('')
ax1.set_title('Probplot against normal distribution')
ax2 = fig.add_subplot(212)
xt, box_cox_lambda = st.boxcox(train_df.price.values)
prob = st.probplot(xt, dist=st.norm, plot=ax2)
ax2.set_title('Probplot after Box-Cox transformation')
plt.show()
```

3.3 Target Distribution - solution



Solution

- Removed 870 records with price as zero
- Performed a box-cox transformation which resulted in a near log transformation of the target

Observation and Future work

- The target distribution is still skewed but not as much as it was before
- Careful selection of loss function

4. Choice of Metric

Most common metrics used in regression are:

- **R2 score**
- Root mean squared error (**RMSE**)
- Mean absolute error (**MAE**)
- Root mean squared log error (**RMSLE**)
- Mean absolute log error (**MALE**)

Since the log transformed target distribution is only slightly skewed, we are good to go with RMSLE

5. Baseline Performance

Baseline performance using Mean: **0.748**
Baseline performance using Median: **0.753**

6. Train Test Split and Cross Validation Strategy

The data needs to be split into train and test data. Since, log transformed target distribution is not very skewed. I used **Random train test split**.

I used **5**-fold cross validation strategy to tune the hyperparameter and used an additional validation set to estimate the performance of the chosen hyperparameters

7. Feature Engineering

1. Label encoding of categorical features
2. Features such as keywords and topics from topic modeling
3. Target mean encoding of high cardinality categorical features

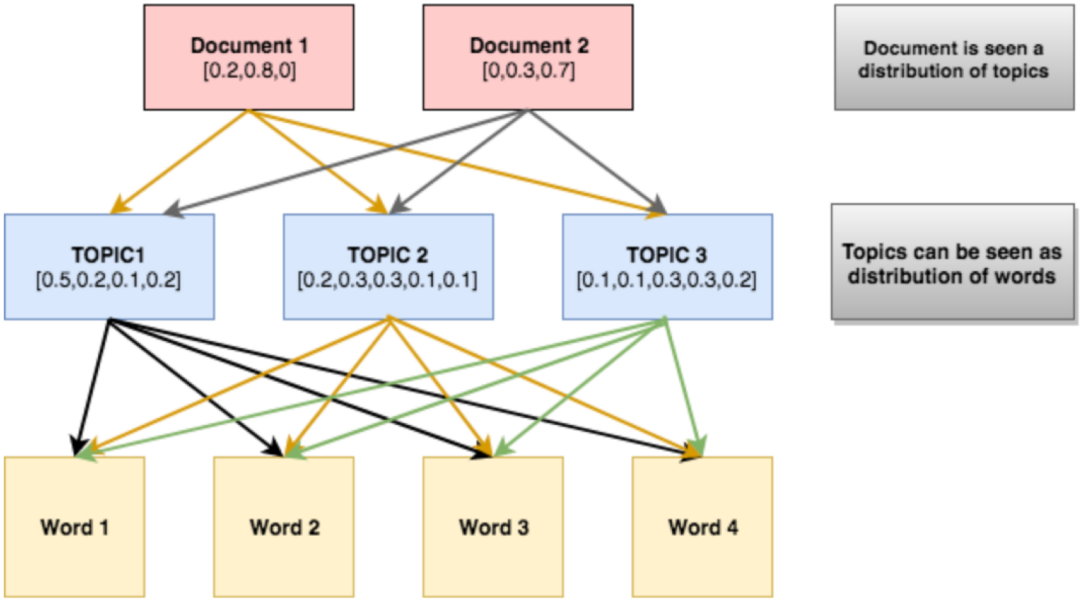
A total of 91 features were developed.

7.1. Label Encoding

- Encoding of categorical columns into codes

```
def train_cats(df):
    '''Change any columns of strings in a panda's dataframe to a column of
       catagorical values. This applies the changes inplace.
    Input:
        df: A pandas dataframe. Any columns of strings will be changed to categorical values.
    '''
    for col_name, col in df.items():
        if is_string_dtype(col): df[col_name] = col.astype('category').cat.as_ordered()


def apply_cats(df, trn):
    '''Changes any columns of strings in df into categorical variables using trn as
       a template for the category codes.
    Input:
        df: A pandas dataframe. Any columns of strings will be changed to categorical values.
        trn: A pandas dataframe. When creating a category for df, it looks up the
            what the category's code were in trn and makes those the category codes
            for df.
    '''

    for n,c in df.items():
        if (n in trn.columns) and (trn[n].dtype.name=='category'):
            df[n] = pd.Categorical(c, categories=trn[n].cat.categories, ordered=True)
```

7.2. Topic Modeling- Overview



Understanding LDA

7.2. Topic Modeling- Data processing pipeline

gensim
topic modelling for humans

POS tagging → Phrase detection → Stop word removal

Bag of words				
	bird	coffee	...	work
doc a	2	0	...	0
doc b	0	1	...	4
...	...	...	...	...
doc D	4	5	...	0

spaCy

7.2. Topic Modeling- Data processing

1. **Data preprocessing**
- Concatenate category name, brand name, name and item description
- Tokenizing text
- Removing verbs, determinants, conjunctions etc with the help of part of speech tagging using Spacy
- Removing stop words (sped up this process by 50% using multicore and multithreading)
- Phrase detection and doc term matrix generation using spa

```
def tokenize_text(df_col):
    text_corpus = df_col.apply(lambda x: x.lower()).values
    removal=['ADV','PRON','CCONJ','PUNCT',
                                    'PART','DET','ADP','SPACE','VERB']
    text_out_final = []

    for review in tqdm(nlp.pipe(text_corpus,n_threads=24,
                                batch_size=10000),desc='Creating tokens'):
        text_out = []
        for token in review:
            if token.pos_ not in removal and token.is_punct == False and token.is_stop == False:
                lemma = token.lemma_
                if lemma != '-PRON-':
                    text_out.append(lemma)

        text_out_final.append(text_out)
    return text_out_final
```

```
from multiprocessing import Pool

def bigramModel(review):
    return bigram_model[review]
bigram_model = Phrases(tokenized_text,scoring='npmi',threshold=0.5,min_count=100)
p = Pool(24)
processed_bigram_reviews = p.map(bigramModel,tokenized_text)
```

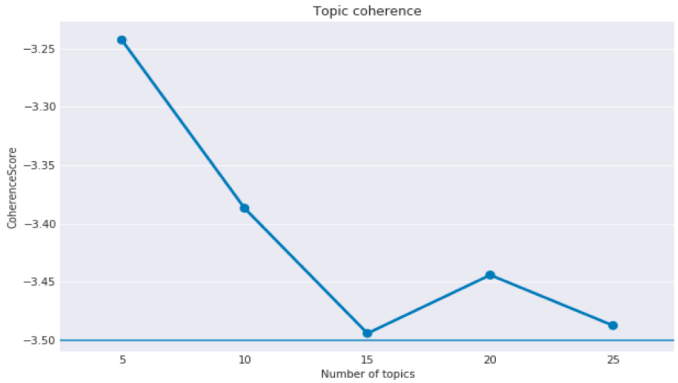
7.2. Topic Modeling – Finding the number of topics

2. **LDA modeling**
- Make the dictionary
- Used topic-coherence plot to find the number of topics

3. **Evaluation of topics using Topic coherence**
The interpretability of each topic is evaluated using a metric called topic coherence

Topic model

```
Lda = models.LdaMulticore
coherenceList_umass = []
num_topics_list = np.arange(5,30,5)
for num_topics in tqdm(num_topics_list):
    lda= Lda(doc_term_matrix, num_topics=num_topics,id2word = dictionary,
             passes=3,chunksize=8000,random_state=43)
    cm = CoherenceModel(model=lda, corpus=doc_term_matrix,
                        dictionary=dictionary, coherence='u_mass')
    coherenceList_umass.append(cm.get_coherence())
    vis = pyLDAvis.gensim.prepare(lda, doc_term_matrix, dictionary)
    pyLDAvis.save_html(vis,f'pyLDAvis_{num_topics}.html')
```



7.2. Topic Modeling – Major topic clusters

Topic	Relevant words
Women clothing	{blouse, woman_top, shirt, size, blouse_t}
Undergarments	{bra, victoria_secret, brush, woman_underwear, pantie}
Phone assessory	{case_skin, phone_accessory, case, iphone, electronic_cell}
Women bags	{woman_handbag, purse, coach, bag, woman}
Gaming	{game, electronic_video, console, game_console, slime}
Phone	{phone_accessory, phone, case, iphone, electronic_cell}
Accessory	{accessory, audio_surveillance, electronic_tv, hair, headphone
Women jewelery	{rm, ring, earring, necklace, woman_jewelry}
Body care	{beauty_skin, care_body, beauty_fragrance, care_face, perfume}
Kid	{0_24, home_décor, baby, home, kid_girl}
Make up	{makeup, beauty_makeup, beauty, face, lip}
woman_athletic	{lularoe, apparel_pant, woman_athletic, legging, tight_legging}

7.2. Topic Modeling – Document to topic vector

Item description	Topic1	Topic2	Topic3
This keyboard is in great condition and works ...	0.6	0.3	0.1
New with tags. Leather horses. Retail for ...	0.2	0.7	0.1

7.3. Regularized target mean encoding

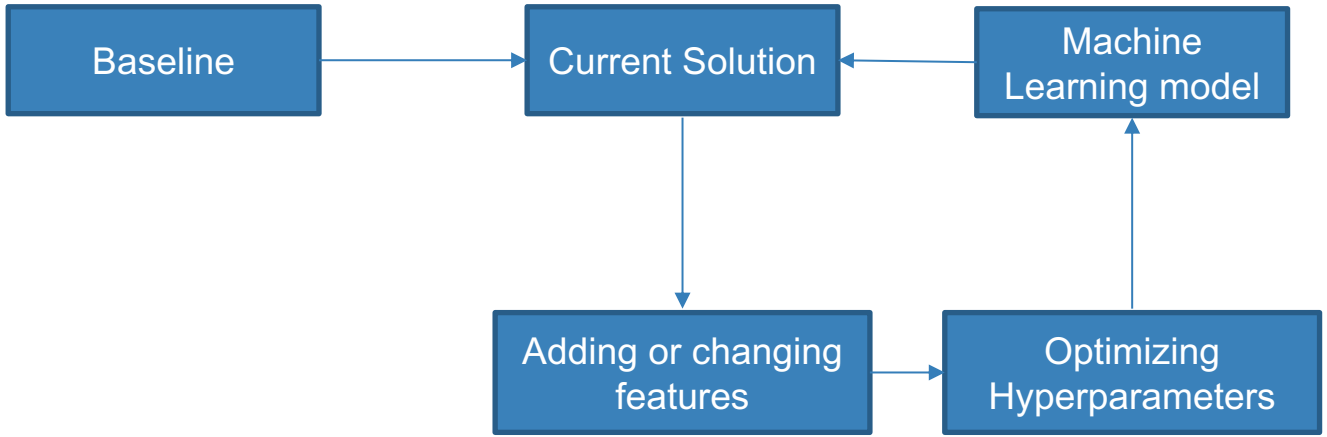
Categorical feature	feature label	target	target mean per feature
iphone	1	125	122.50
iphone	1	120	122.50
iphone	1	115	122.50
iphone	1	130	122.50
Samsung	2	80	96.25
Samsung	2	110	96.25
Samsung	2	95	96.25
Samsung	2	100	96.25
Mi phone	3	50	60.00
Mi phone	3	60	60.00
Mi phone	3	70	60.00
Blackberry	4	35	33.33
Blackberry	4	40	33.33
Blackberry	4	25	33.33

```
def reg_target_encoding(train, col,target_col, splits=5):
    """ Computes regularize mean encoding.
    Inputs:
        train: training dataframe
        col : column name on which you want to perfrom mean encoding
        target_col : name of the target column
    Output:
        train: training data with regualarized mean encoded features
    """
    kf = KFold(n_splits=splits)
    global_mean = train[target_col].mean()
    for train_index,test_index in kf.split(train):
        kfold_mean = train.iloc[train_index,:].groupby(col)[target_col].mean()
        train.loc[test_index,col+'_mean_enc'] =  train.loc[test_index,col].map(kfold_mean)
    train[col+"_mean_enc"].fillna(global_mean, inplace=True)
    train[col+"_mean_enc"] = train[col+"_mean_enc"].astype('float32')
    return train

def mean_encoding_test(test, train, col,target_col):
    """ Computes target encoding for test data.
    Inputs:
        test: test dataframe
        train: training dataframe
        col: column name on which you want to perfrom mean encoding
        target_col : name of the target column
    Outputs:
        train: training data with regualarized mean encoded features
    """
    global_mean = train[target_col].mean()
    mean_col = train.groupby(col)[target_col].mean()
    test[col+"_mean_enc"] = test[col].map(mean_col)
    test[col+"_mean_enc"].fillna(global_mean, inplace=True)
    test[col+"_mean_enc"] = test[col+"_mean_enc"].astype('float32')
    return test
```

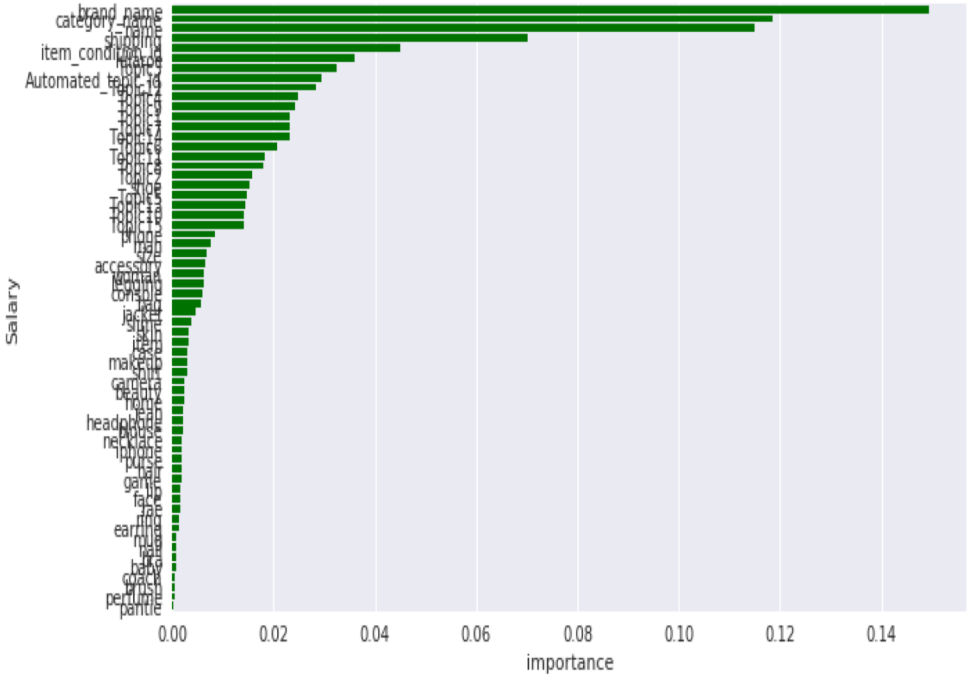
8. Learning Algorithms

1. Random Forest Regressor
2. XGBoost Regressor
3. 1-D Convolutional Neural Network + Categorical embeddings

8.1 Learning Algorithms – RF and XGBoost

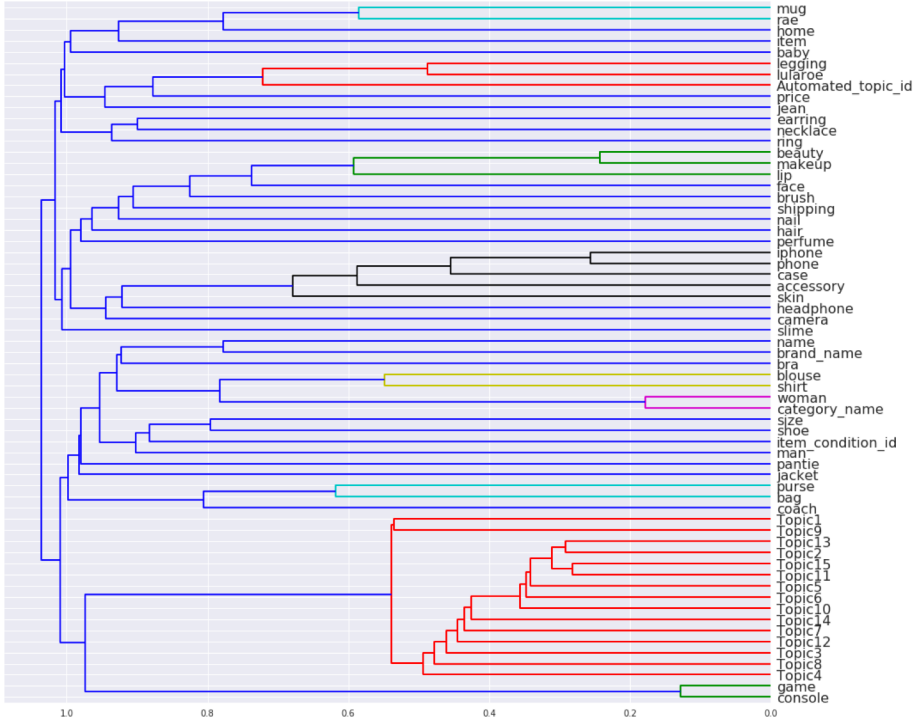


8.1.1 Adding or changing features



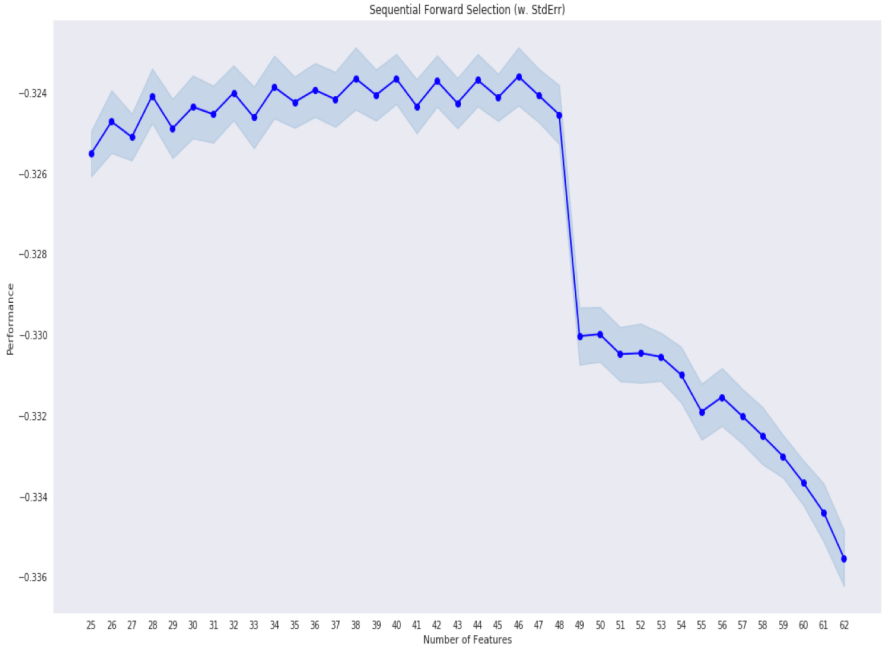
- Started with Random forest with 91 features
- Removed features with zero importance

8.1.1 Adding or changing features



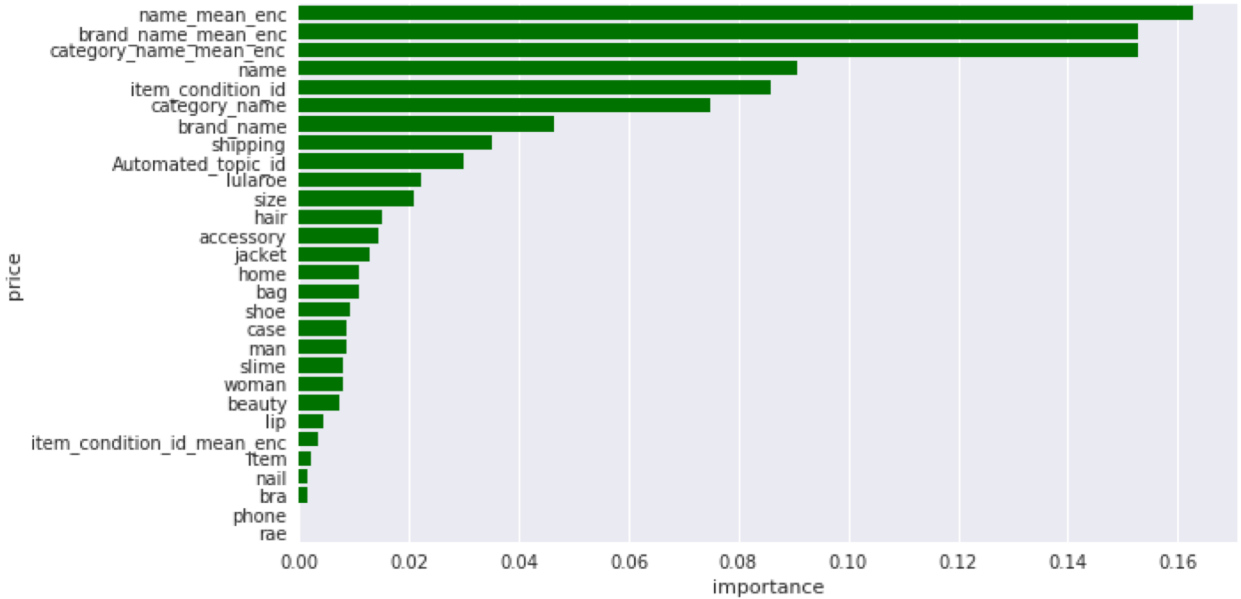
- Started with Random forest with 91 features
- Removed features with zero importance
- Identify collinear features with dendrogram and removed one of them.

8.1.1 Adding or changing features



- Started with Random forest with 91 features
- Removed features with zero importance
- Identify collinear features with dendrogram and removed one of them.
- The number of features dropped to 62
- Used backward elimination process to find a subset of 25 important parameters from 62 without drop in performance

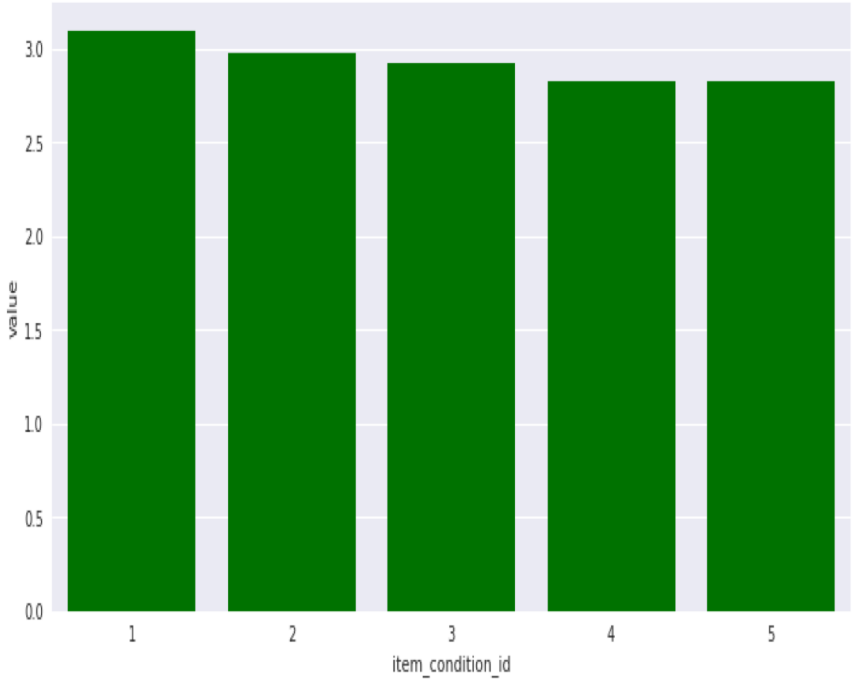
8.1.1 Adding or changing features: Final features



9.1 Final results

Machine Learning Algorithm	RMSLE on Validation
Random Forest	0.556
XGBoost	0.501

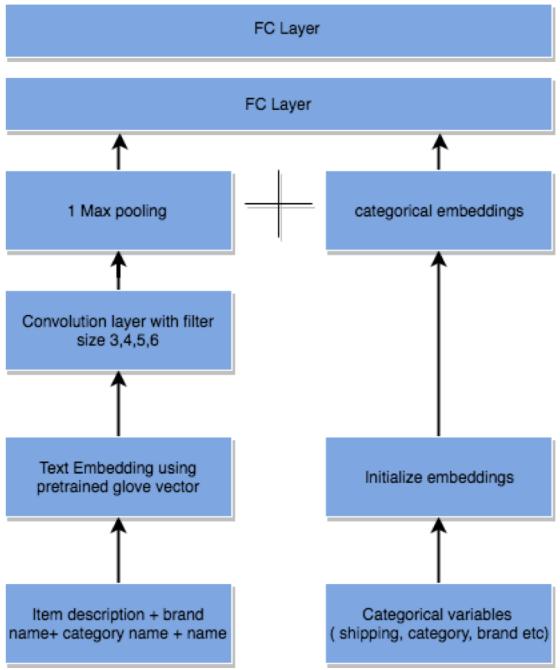
9.2 Results- Partial Dependence plot

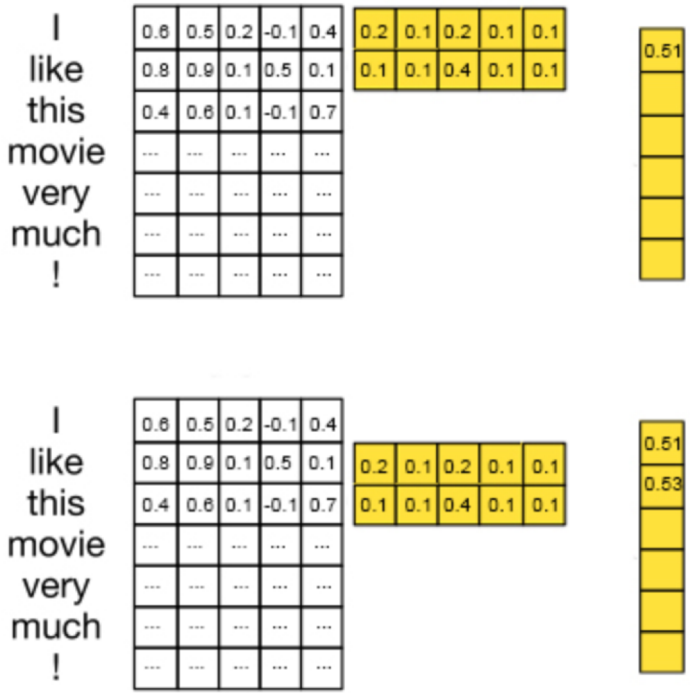


- In the problem statement, it was not given what each item condition meant.
- With the help of partial dependence plot. We are able to ascertain which item condition id is favorable and which isn't.

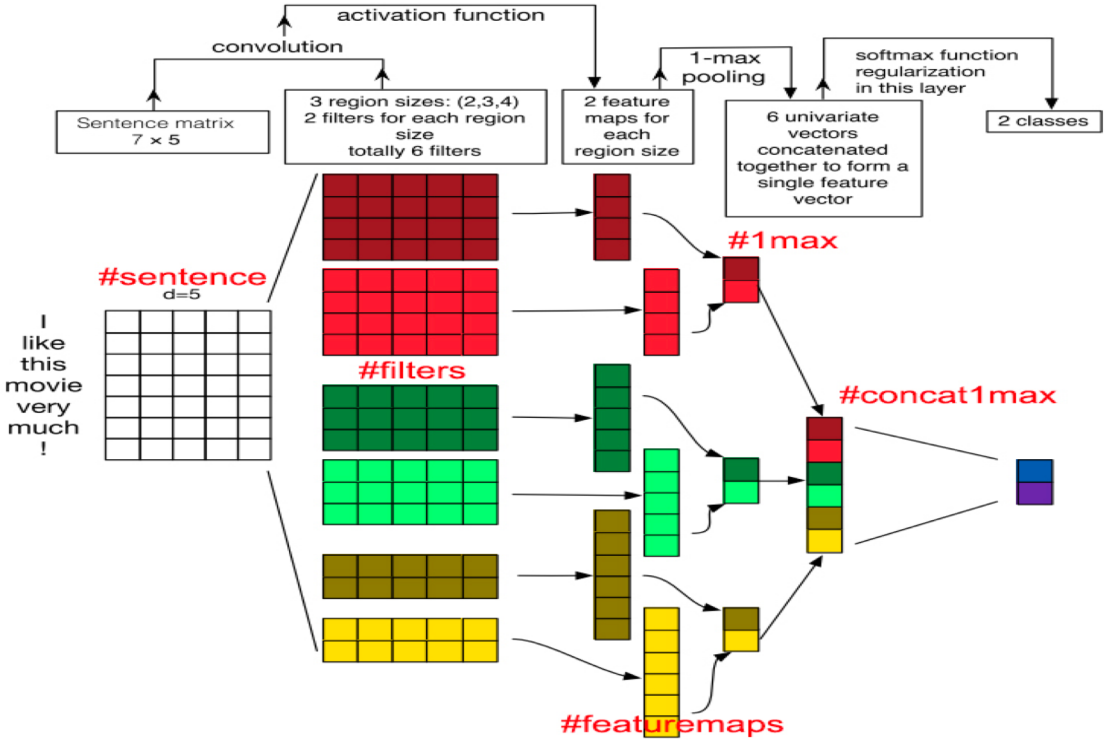
```
def partial_dependence(X, fldname,m):
    ''' Computes partial dependence or the change in target with respect to the change in the selected feature
    Inputs:
        X: dataframe with features
        fldname: selected column
        m: trained model
        categorymap: maps each category with its id
    Output:
        final: dataframe that has partial dependence value for each value
    '''

    Xnew = X.copy()
    xreq = X[fldname].sort_values().unique()
    ypred = np.zeros((Xnew.shape[0],len(xreq)))
    counter =0
    for i in xreq:
        Xnew[fldname] = i
        ypred[:,counter] = m.predict(Xnew)
        counter +=1
    final = pd.DataFrame(ypred,columns=[xreq]).apply(np.mean,axis=0).to_frame()
    final.reset_index(inplace=True)
    final['level_0']=final['level_0']
    final.columns = [fldname,'value']
    return final
```

Bonus: 1-D CNN Architecture





Bonus: 1-D CNN Intuition



9.1 Final results

Learning Algorithm	RMSLE on Validation
Random Forest	0.556
XGBoost	0.501
1-D CNN	0.421

Thanks!

Any questions ?