In [ ]:

```
# Link for Downloading "best_model.h5" and "best_model_cl.json".
#https://drive.google.com/open?id=1-0L1P3O4s0Hv-lL-ZWXZUlbw6U7dTy7Q
#https://drive.google.com/open?id=1-4UqEmlqCB4G0m-W1uvf4otDFGiUf0sk
```

In [2]:

```
# Importing Libraries.
from datetime import datetime
start_real = datetime.now()
import numpy as np
import pandas as pd
import math
import pickle
from nltk.corpus import stopwords
from tqdm import tqdm
from keras.models import model_from_json

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import RidgeCV
from sklearn.pipeline import FeatureUnion

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Input, Dropout, Dense, concatenate
from keras.layers import GRU, Embedding, Flatten, Activation
from keras.models import Model

from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

from keras.layers import Input, Dense, Embedding, Dense, Dropout, Flatten, Conv1D, Glob
alMaxPooling1D, BatchNormalization, LSTM, GRU
from keras.models import Model
from keras import optimizers
```

```
Using TensorFlow backend.
```

In [0]:

```
# Defining Utility Functions :

# 1. RMSLE Function
def rmsle(y, y_pred): # return Rmsle value.
    return np.sqrt(np.mean(np.square(y_pred - y )))

#--------------------------------------------------------------------

# 2. Word Count Function.
def word_count(text):
    try:
        if text == 'No description yet':
            return 0 # for the data point with string "No description yet" returns word
count 0.
        else:
            text = text.lower()
            words = []
            for w in text.split(" "):
              words.append(w)
            return len(words)
    except:
        return 0

#--------------------------------------------------------------------


# 3.Splitting category into sub categories.
def cat_split(column, sub_cat):
    category = []
    for i in range(len(train)):
        try:
            category.append(train[column].values[i].split("/")[sub_cat])
        except:
            category.append("No Label") # If there is no sub category it repalces No La
bel.

    return category

def cat_split_test(column, sub_cat, test):
    category = []
    for i in range(len(test)):
        try:
            category.append(test[column].values[i].split("/")[sub_cat])
        except:
            category.append("No Label") # If there is no sub category it repalces No La
bel.
    return category

#--------------------------------------------------------------------


# 4. finding missing brands.
#https://www.kaggle.com/valkling/mercari-rnn-2ridge-models-with-notes-0-42755
# The Brand Name has 600,000 Missing Values. This Function will replace the data.
def finding_brand(row_n):
    brand_row = row_n[0]
    name = row_n[1]
    namesplit = name.split(' ') # for missing brand we check every word in the name col
umn
```

```
    if brand_row == 'missing':
        for x in namesplit: # Then we check for every word in brand vocabulary.if exist
s retur name
            if x in brand_vocab:
                return name
    if name in brand_vocab:
        return name
    return brand_row

#--------------------------------------------------------------------

# 5. Filling Missing values. Filling Columns names, category_name, item_description, br
and_name
def fill_missing_values(df):
    df.category_name.fillna(value="missing", inplace=True)
    df.brand_name.fillna(value="missing", inplace=True)
    df.item_description.fillna(value="missing", inplace=True)
    df.item_description.replace('No description yet',"missing", inplace=True)
    return df

# https://stackoverflow.com/a/56876351
from sklearn.preprocessing import LabelEncoder
import numpy as np


class LabelEncoderExt(object):
    def __init__(self):
        """
        It differs from LabelEncoder by handling new classes and providing a value for
 it [Unknown]
        Unknown will be added in fit and transform will take care of new item. It gives
unknown class id
        """
        self.label_encoder = LabelEncoder()
        # self.classes_ = self.label_encoder.classes_

    def fit(self, data_list):
        """
        This will fit the encoder for all the unique values and introduce unknown value
        :param data_list: A list of string
        :return: self
        """
        self.label_encoder = self.label_encoder.fit(list(data_list) + ['Unknown'])
        self.classes_ = self.label_encoder.classes_

        return self

    def transform(self, data_list):
        """
        This will transform the data_list to id list where the new values get assigned
 to Unknown class
        :param data_list:
        :return:
        """
        new_data_list = list(data_list)
        for unique_item in np.unique(data_list):
            if unique_item not in self.label_encoder.classes_:
                new_data_list = ['Unknown' if x==unique_item else x for x in new_data_l
ist]

        return self.label_encoder.transform(new_data_list)
```

In [0]:

```
def final_fun_1(data_point):
  test = pd.DataFrame(data_point).T
  test.columns = ['id', 'name', 'item_condition_id', 'category_name', 'brand_name','pri
ce', 'shipping', 'item_description']
  test['desc_len'] = test['item_description'].apply(lambda x: word_count(x))
  test['name_len'] = test['name'].apply(lambda x: word_count(x))
  test["subcat_0"] = cat_split_test("category_name",0, test)
  test["subcat_1"] = cat_split_test("category_name",1, test)
  test["subcat_2"] = cat_split_test("category_name",2, test)
  global brand_vocab
  brand_vocab = set(test['brand_name'].values)
  test.brand_name.fillna(value = "missing", inplace = True)
  missing = len(test.loc[test["brand_name"] == 'missing'])
  test['brand_name'] = test[['brand_name','name']].apply(finding_brand, axis = 1)
  detected_brands = missing-len(test.loc[test['brand_name'] == 'missing'])
  test = fill_missing_values(test)
  test["combined_text"] = test["item_description"] + " " + test["name"]
  test['brand_name'] = test['brand_name'].fillna('missing').astype(str)
  label = LabelEncoderExt()
  label.fit(np.hstack([test.brand_name]))
  test['brand_name'] = label.transform(test.brand_name)
  del label
  label = LabelEncoderExt()
  label.fit(np.hstack([test.category_name])) # categories united
  test['category'] = label.transform(test.category_name)
  label.fit(np.hstack([test.subcat_0])) # sub_cat0
  test.subcat_0 = label.transform(test.subcat_0)
  label.fit(np.hstack([test.subcat_1])) # sub_cat_1
  test.subcat_1 = label.transform(test.subcat_1)
  label.fit(np.hstack([test.subcat_2])) # sub_cat2
  test.subcat_2 = label.transform(test.subcat_2)
  del label
  full_text = np.hstack([test.item_description.str.lower(), test.name.str.lower(), test
.category_name.str.lower()])
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(full_text)
  test["seq_combined"] = tokenizer.texts_to_sequences(test.combined_text.str.lower())
  test['seq_desc'] = tokenizer.texts_to_sequences(test.item_description.str.lower())
  test['seq_name'] = tokenizer.texts_to_sequences(test.name.str.lower())
  max_len_combined = np.max([test.seq_combined.max()]) + 1
  max_len_brand =  np.max([test.brand_name.max()]) + 1   # brand # brand
  max_len_condition = np.max([test.item_condition_id]) + 1  # item_cond
  max_len_desc = np.max([int(int(test.desc_len.max()))]) + 1  # item_desc_len
  max_len_name = np.max([int(int(test.name_len.max()))]) + 1 # name_len
  max_len_sub0 = np.max([int(int(test.subcat_0.max()))])  + 1# Sub_0
  max_len_sub1 = np.max([int( int(test.subcat_1.max()))]) + 1 # Sub_1
  max_len_sub2 = np.max([ test.subcat_2.max()]) + 1  # Sub_2
  name_padding = 10
  description_padding = 70
  combined_padding  =  70
  max_len = np.max([np.max(test.seq_name.max()),np.max(test.seq_desc.max())]) + 1
  te_data = {
  "name" : pad_sequences(test.seq_name, maxlen= name_padding),
  "item_desc" : pad_sequences(test.seq_desc, maxlen= description_padding),
  "brand_name" : np.array(test.brand_name),
  "category" : np.array(test.category),
  "item_condition" : np.array(test.item_condition_id),
  "shipping" : np.array(test[["shipping"]]),
  "desc_len" : np.array(test[["desc_len"]]),
```

```
"name_len" : np.array(test[["name_len"]]),
"subcat_0" : np.array(test.subcat_0),
"subcat_1" : np.array(test.subcat_1),
"subcat_2" : np.array(test.subcat_2),
"combined_text" : pad_sequences(test.seq_combined, maxlen= combined_padding)
}
X_test = te_data
json_file = open('/content/drive/My Drive/best_model_cl.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("/content/drive/My Drive/best_model.h5")
y_pred = loaded_model.predict(X_test)
test_pred = np.expm1(y_pred)
return test_pred
```

In [0]:

```
def final_fun_2(data_point, target):
  test = pd.DataFrame(data_point).T
  test.columns = ['id', 'name', 'item_condition_id', 'category_name', 'brand_name','pri
ce', 'shipping', 'item_description']
  y = np.log1p(target)
  test['desc_len'] = test['item_description'].apply(lambda x: word_count(x))
  test['name_len'] = test['name'].apply(lambda x: word_count(x))
  test["subcat_0"] = cat_split_test("category_name",0, test)
  test["subcat_1"] = cat_split_test("category_name",1, test)
  test["subcat_2"] = cat_split_test("category_name",2, test)
  global brand_vocab
  brand_vocab = set(test['brand_name'].values)
  test.brand_name.fillna(value = "missing", inplace = True)
  missing = len(test.loc[test["brand_name"] == 'missing'])
  test['brand_name'] = test[['brand_name','name']].apply(finding_brand, axis = 1)
  detected_brands = missing-len(test.loc[test['brand_name'] == 'missing'])
  test = fill_missing_values(test)
  test["combined_text"] = test["item_description"] + " " + test["name"]
  test['brand_name'] = test['brand_name'].fillna('missing').astype(str)
  label = LabelEncoderExt()
  label.fit(np.hstack([test.brand_name]))
  test['brand_name'] = label.transform(test.brand_name)
  del label
  label = LabelEncoderExt()
  label.fit(np.hstack([test.category_name])) # categories united
  test['category'] = label.transform(test.category_name)
  label.fit(np.hstack([test.subcat_0])) # sub_cat0
  test.subcat_0 = label.transform(test.subcat_0)
  label.fit(np.hstack([test.subcat_1])) # sub_cat_1
  test.subcat_1 = label.transform(test.subcat_1)
  label.fit(np.hstack([test.subcat_2])) # sub_cat2
  test.subcat_2 = label.transform(test.subcat_2)
  del label
  full_text = np.hstack([test.item_description.str.lower(), test.name.str.lower(), test
.category_name.str.lower()])
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(full_text)
  test["seq_combined"] = tokenizer.texts_to_sequences(test.combined_text.str.lower())
  test['seq_desc'] = tokenizer.texts_to_sequences(test.item_description.str.lower())
  test['seq_name'] = tokenizer.texts_to_sequences(test.name.str.lower())
  max_len_combined = np.max([test.seq_combined.max()]) + 1
  max_len_brand =  np.max([test.brand_name.max()]) + 1   # brand # brand
  max_len_condition = np.max([test.item_condition_id]) + 1  # item_cond
  max_len_desc = np.max([int(int(test.desc_len.max()))]) + 1  # item_desc_len
  max_len_name = np.max([int(int(test.name_len.max()))]) + 1 # name_len
  max_len_sub0 = np.max([int(int(test.subcat_0.max()))])  + 1# Sub_0
  max_len_sub1 = np.max([int( int(test.subcat_1.max()))]) + 1 # Sub_1
  max_len_sub2 = np.max([ test.subcat_2.max()]) + 1  # Sub_2
  name_padding = 10
  description_padding = 70
  combined_padding  =  70
  max_len = np.max([np.max(test.seq_name.max()),np.max(test.seq_desc.max())]) + 1
  te_data = {
  "name" : pad_sequences(test.seq_name, maxlen= name_padding),
  "item_desc" : pad_sequences(test.seq_desc, maxlen= description_padding),
  "brand_name" : np.array(test.brand_name),
  "category" : np.array(test.category),
  "item_condition" : np.array(test.item_condition_id),
  "shipping" : np.array(test[["shipping"]]),
```

```
  "desc_len" : np.array(test[["desc_len"]]),
  "name_len" : np.array(test[["name_len"]]),
  "subcat_0" : np.array(test.subcat_0),
  "subcat_1" : np.array(test.subcat_1),
  "subcat_2" : np.array(test.subcat_2),
  "combined_text" : pad_sequences(test.seq_combined, maxlen= combined_padding)
  }
  X_test = te_data
  json_file = open('/content/drive/My Drive/best_model_cl.json', 'r')
  loaded_model_json = json_file.read()
  json_file.close()
  loaded_model = model_from_json(loaded_model_json)
  # load weights into new model
  loaded_model.load_weights("/content/drive/My Drive/best_model.h5")
  y_pred = loaded_model.predict(X_test)
  #test_pred = np.expm1(y_pred)
  score = rmsle(y, y_pred)
  return score
```

In [11]:

```
# Testing  Both the functions.
train = pd.read_table('/content/drive/My Drive/train.tsv')
data_point = list(train.loc[10])
price = train['price'].loc[10]

# We get sample data point from Train data and see the actual price and predicted price
and rmsle score.
print("The actal price :", price ,",Predicted price:", final_fun_1(data_point))
print("Rmsle Score:",final_fun_2(data_point,price) )
```

```
The actal price : 8.0 ,Predicted price: [[9.847242]]
Rmsle Score: 0.18668628
```