Problem Statement.

Given a comment classify it into toxic or non-toxic. Here toxic means comments that are unacceptable to a person or a community.

This project was inspired from the jigsaw unintended bias toxic comment classification which was hosted on kaggle.

The problem that we are solving here.

Background :

This problem was also solved earlier but there was a problem with the system . The system was a bit biased towards the use of unparliamentary/abusive words.

For example a sentence "I am a gay" or "I am a black man." were classified as toxic comments.

Therefore the challenge was to get rid of this bias. To solve this problem a new metric was introduced by jigsaw. This metric also considers the identity of the person about whom the comment is made. You can get into the details of the metric on the below link.

Click Here

Exploratory Data Analysis

In [67]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [68]:

```
import pandas as pd
df = pd.read_csv("/content/drive/My Drive/Project/train.csv")
df.head()
```

Out[68]:

	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat	asian	atheist	bisexual	black
0	59848	0.000000	This is so cool. It's like, 'would you want yo...	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	NaN	NaN
1	59849	0.000000	Thank you!! This would make my life a lot less...	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	NaN	NaN
2	59852	0.000000	This is such an urgent design problem; kudos t...	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	NaN	NaN
3	59855	0.000000	Is this something I'll	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	NaN	NaN

	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat	asian	atheist	bisexual	black
3	59855	0.000000	install on m...	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	NaN	NaN
4	59856	0.893617	haha you guys are a bunch of losers.	0.021277	0.0	0.021277	0.87234	0.0	0.0	0.0	0.0	0.0

In [69]:

```
#print(df.shape)
#df = df.sample(500000)
print(df.shape)
```

```
(1804874, 45)
```

In [70]:

```
df = df.drop_duplicates(subset={'comment_text'},keep='first')
df.shape
```

Out[70]:

```
(1780823, 45)
```

Looking the absuive words used in the comments.

In [71]:

```
# Lets find the words that contains letters or character that are not present in english
import re
from tqdm import tqdm

non_eng_words = {}
pattern = re.compile(r'([fadpbit]{1,4}[*]{1,4}[ingkscyhabd]{1,4})')

for sent in tqdm(df['comment_text'].values):
    sent = re.sub(r'[^a-zA-Z* ]',' ',sent)
    for wrd in sent.split():
        if(re.match(pattern,wrd) and len(wrd)>2):
            if(non_eng_words.get(wrd,-1)<0):
                non_eng_words[wrd] = 1
            else:
                non_eng_words[wrd] += 1
```

```
100%|██████████| 1780823/1780823 [02:01<00:00, 14638.05it/s]
```

In [72]:

```
# Now we will store these words and their counts in a list.
wrds = []
counts = []
for wrd in sorted(non_eng_words, key=non_eng_words.get, reverse=True):
    if(len(wrd)<30):
        wrds.append(wrd)
        counts.append(non_eng_words[wrd])
print("Number of aabusive words wused are ".format(len(non_eng_words)))

## Abusive words present in the comments.
for i in range(len(wrds)):
    print(wrds[i],"->",counts[i])
```

```
Number of aabusive words wused are
p***y -> 73
p*ssy -> 65
f**k -> 45
a**hole -> 38
f*ck -> 29
a**holes -> 27
p*ss -> 25
```

```
f*cking -> 18
f***ing -> 17
f**king -> 15
p*ssed -> 13
p**sy -> 13
a*s -> 10
f**ked -> 10
b***h -> 9
f*cked -> 8
d*ck -> 7
p**s -> 7
p*ssie -> 6
p*ssing -> 6
a**clown -> 6
bit*h -> 5
d**k -> 5
b**ch -> 5
f*cks -> 4
p**y -> 4
d****d -> 3
p***ing -> 3
pi*s -> 3
p***ies -> 3
p*nis -> 3
b*itch -> 3
d**n -> 3
d**ks -> 3
b*stards -> 3
f**cker -> 3
p****y -> 3
a**clowns -> 3
p****d -> 2
pi**ing -> 2
bi*ch -> 2
a*sholes -> 2
f**ks -> 2
f*k -> 2
a*se -> 2
f***in -> 2
f***n -> 2
d*amn -> 2
f**kin -> 2
p*ss* -> 2
p*ssies -> 2
f****d -> 2
f****ing -> 2
a**hats -> 2
b***s -> 2
p*sses -> 2
p*ss*ng -> 2
f***king -> 2
b*stard -> 2
p*s*y -> 2
f*ckers -> 2
d*ckhead -> 2
f****s -> 2
a***holes -> 2
a*holes -> 2
t**d -> 2
f*ckin -> 1
b**ching -> 1
a*a* -> 1
f*c*ing -> 1
bat*hit -> 1
f***g -> 1
d***s -> 1
a*shole -> 1
p****s -> 1
b*sta*ds -> 1
p**s* -> 1
p****ies -> 1
b*gg*r -> 1
f*c*king -> 1
f***s -> 1
a*ss -> 1
bat****crazy -> 1
p*sspot -> 1
```

```
p***ys -> 1
ba**ards -> 1
f**kc -> 1
di*k -> 1
f*ck*d -> 1
b*alls -> 1
di*ks -> 1
f*c* -> 1
b****h -> 1
f*kcyouCanada -> 1
t*g -> 1
a**h*** -> 1
f**ken -> 1
f**g -> 1
f**ing -> 1
a**holery -> 1
f**kem -> 1
i*iot -> 1
a**hat -> 1
f**cking -> 1
f**ching -> 1
d*icks -> 1
t*i*t -> 1
a**h**e -> 1
f***ked -> 1
b***s**t -> 1
b*st*rd -> 1
b***ch -> 1
p*as -> 1
a****and -> 1
f*ggots -> 1
da*n -> 1
a**h* -> 1
f**cked -> 1
t*h -> 1
id*iot -> 1
b****s -> 1
bit*hes -> 1
b***ches -> 1
fa**in -> 1
b**hes -> 1
da*ned -> 1
p**a -> 1
t**s -> 1
di*cks -> 1
d**ned -> 1
f*ckup -> 1
a***hole -> 1
p*sssy -> 1
a*hole -> 1
f***k -> 1
p****grabber -> 1
f***able -> 1
bi**h -> 1
a**ing -> 1
d*do -> 1
p**ck -> 1
di**s -> 1
f*ggot -> 1
d*ke -> 1
f*g -> 1
```

Data Cleaning

In [0]:

```
# Ref: https://www.kaggle.com/haqishen/jigsaw-predict
# We are creating a dict with shortened word as key and actual word as value.
apostophe_dict = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because", "could've": "could have",
                  "couldn't": "could not", "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't": "had not",
                  "hasn't": "has not", "haven't": "have not", "he'd": "he would","he'll": "he will", "he's": "he is",
                  "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's": "how
```

```
                    "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's": "how
is",  "I'd": "I would",
                    "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have","I'm": "I
am", "I've": "I have",
                    "i'd": "i would", "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will
have","i'm": "i am",
                    "i've": "i have", "isn't": "is not", "it'd": "it would", "it'd've": "it would hav
", "it'll": "it will",
                    "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "madam",
"mayn't": "may not",
                    "might've": "might have","mightn't": "might not","mightn't've": "might not have",
"must've": "must have",
                    "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "nee
dn't've": "need not have",
                    "o'clock": "of the clock", "oughtn't": "ought not", "oughtn't've": "ought not hav
e", "shan't": "shall not",
                    "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she would",
"she'd've": "she would have",
                    "she'll": "she will", "she'll've": "she will have", "she's": "she is",
"should've": "should have",
                    "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have",
"so's": "so as",
                    "this's": "this is","that'd": "that would", "that'd've": "that would have",
"that's": "that is",
                    "there'd": "there would", "there'd've": "there would have", "there's": "there is"
, "here's": "here is",
                    "they'd": "they would", "they'd've": "they would have", "they'll": "they will",
"they'll've": "they will have",
                    "they're": "they are", "they've": "they have", "to've": "to have", "wasn't": "was
not", "we'd": "we would",
                    "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we'r
": "we are",
                    "we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've":
"what will have",
                    "what're": "what are",  "what's": "what is", "what've": "what have", "when's":
"when is",
                    "when've": "when have", "where'd": "where did", "where's": "where is", "where've"
: "where have",
                    "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've":
"who have",
                    "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will n
ot",
                    "won't've": "will not have", "would've": "would have", "wouldn't": "would not",
                    "wouldn't've": "would not have", "y'all": "you all", "y'all'd": "you all would",
                    "y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all
have","you'd": "you would",
                    "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have",
"you're": "you are",
                    "you've": "you have" }
```

In [74]:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords = list(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [0]:

```
import re

def process_sent(sent):
    line = ''
    for wrd in sent.split():
        if(len(apostophe_dict.get(wrd.lower(),'n'))>1 and wrd.lower() not in stopwords and len(wrd)
>2):
            wrd = apostophe_dict[wrd.lower()]
            n_wrd = ''
            for w in wrd.split():
                if(w not in stopwords):
                    n_wrd += " "+w
            line += " "+n_wrd
```

```
        else:
            line += " "+ wrd.lower()
    line = re.sub(r'[^a-zA-Z* ]',' ',line)
    return line
```

In [104]:

```
# Preprocessing the comments.
preprocessed_data = []
from tqdm import tqdm
for sent in tqdm(df['comment_text'].values):
    sent = process_sent(sent)
    line = ''
    for wrd in sent.split():
        if(len(wrd)>2):
            line += " " +wrd.lower()
    line = re.sub(r"[']","',line)
    preprocessed_data.append(line)
```

```
100%|██████████| 1780823/1780823 [01:28<00:00, 20101.91it/s]
```

In [0]:

```
df['comment_text'] = preprocessed_data
```

Splitting data

In [78]:

```
from sklearn.model_selection import train_test_split
Y = df['target'].values
train,test,Y_train,Y_test = train_test_split(df,Y,test_size=0.2,random_state=42)
train,cv,Y_train,Y_cv = train_test_split(train,Y_train,test_size=0.1,random_state=42)
print((train.shape),(test.shape),(cv.shape))
```

```
(1282192, 45) (356165, 45) (142466, 45)
```

In [0]:

```
identity_columns = [
        'male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',
        'muslim', 'black', 'white', 'psychiatric_or_mental_illness']
```

In [0]:

```
import numpy as np
seed = 1029

train_x = train['comment_text'].fillna('_##_').values
test_x = test['comment_text'].fillna('_##_').values
cv_x = cv['comment_text'].fillna('_##_').values

# For gtrain
weights = np.ones((len(train),))
weights += train[identity_columns].fillna(0).values.sum(axis=1) * 3
weights += train['target'].values * 8
weights /= weights.max()
train_y = np.vstack([train['target'], weights]).T
train_y_identity = train[identity_columns].values

# For test
weights = np.ones((len(test),))
weights += test[identity_columns].fillna(0).values.sum(axis=1) * 3
weights += test['target'].values * 8
weights /= weights.max()
test_y = np.vstack([test['target'], weights]).T
test_y_identity = test[identity_columns].values

# For cv
weights = np.ones((len(cv),))
```

```
weights = np.ones((len(cv),))
weights += cv[identity_columns].fillna(0).values.sum(axis=1) * 3
weights += cv['target'].values * 8
weights /= weights.max()
cv_y = np.vstack([cv['target'], weights]).T
cv_y_identity = cv[identity_columns].values

# shuffling the data
np.random.seed(seed)
train_idx = np.random.permutation(len(train_x))
test_idx = np.random.permutation(len(test_x))
cv_idx = np.random.permutation(len(cv_x))

train_x = train_x[train_idx]
train_y = train_y[train_idx]
train_y_identity = train_y_identity[train_idx]


test_x = test_x[test_idx]
test_y = test_y[test_idx]
test_y_identity = test_y_identity[test_idx]


cv_x = cv_x[cv_idx]
cv_y = cv_y[cv_idx]
cv_y_identity = cv_y_identity[cv_idx]
```

In [81]:

```
y_binary = (test_y[:, 0] >= 0.5).astype(int)
y_identity_binary = (test_y_identity >= 0.5).astype(int)
y_identity_binary.shape
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: RuntimeWarning: invalid value
encountered in greater_equal
```

Out[81]:

```
(356165, 9)
```

In [82]:

```
Y_train = (train_y[:,0]>=0.5).astype(int)
Y_cv = (cv_y[:,0]>=0.5).astype(int)
Y_test = (test_y[:,0]>=0.5).astype(int)
print(Y_train.shape, Y_cv.shape)
```

```
(1282192,) (142466,)
```

Custom metric given by *Kaggle*

In [0]:

```
# This metric code below was taken from kaggle
from sklearn.metrics import roc_auc_score
import keras.backend as K
class JigsawEvaluator:

    def __init__(self, y_binary, y_identity_binary, power=-5, overall_model_weight=0.25):
        self.y = y_binary
        self.y_i = y_identity_binary
        self.n_subgroups = self.y_i.shape[1]
        self.power = power
        self.overall_model_weight = overall_model_weight

    @staticmethod
    def _compute_auc(y_true, y_pred):
        try:
            return roc_auc_score(y_true, y_pred)
        except ValueError:
            return np.nan
```

```
    def _compute_subgroup_auc(self, i, y_pred):
        mask = self.y_i[:, i] == 1
        return self._compute_auc(self.y[mask], y_pred[mask])

    def _compute_bpsn_auc(self, i, y_pred):
        mask = self.y_i[:, i] + self.y == 1
        return self._compute_auc(self.y[mask], y_pred[mask])

    def _compute_bnsp_auc(self, i, y_pred):
        mask = self.y_i[:, i] + self.y != 1
        return self._compute_auc(self.y[mask], y_pred[mask])

    def compute_bias_metrics_for_model(self, y_pred):
        records = np.zeros((3, self.n_subgroups))
        for i in range(self.n_subgroups):
            records[0, i] = self._compute_subgroup_auc(i, y_pred)
            records[1, i] = self._compute_bpsn_auc(i, y_pred)
            records[2, i] = self._compute_bnsp_auc(i, y_pred)
        return records

    def _calculate_overall_auc(self, y_pred):
        return roc_auc_score(self.y, y_pred)

    def _power_mean(self, array):
        total = sum(np.power(array, self.power))
        return np.power(total / len(array), 1 / self.power)

    def get_final_metric(self, y_pred):
        #y_pred = K.flatten(y_pred)
        bias_metrics = self.compute_bias_metrics_for_model(y_pred)
        bias_score = np.average([
            self._power_mean(bias_metrics[0]),
            self._power_mean(bias_metrics[1]),
            self._power_mean(bias_metrics[2])
        ])
        overall_score = self.overall_model_weight * self._calculate_overall_auc(y_pred)
        bias_score = (1 - self.overall_model_weight) * bias_score
        return overall_score + bias_score
```

Vectorizing datset using tfidf

vectorizing train dataset

In [0]:

```
# target with value greater than equal to 0.5 will be assigned 1 and rest 0.
def fun(x):
    if(x>=0.5):
        return 1
    else:
        return 0
```

In [0]:

```
labels = df['target']
Y = labels.map(fun)
```

In [107]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(preprocessed_data,Y,test_size=0.2,random_state=42)
X_train,X_cv,Y_train,Y_cv = train_test_split(X_train,Y_train,test_size=0.1,random_state=42)
print(len(X_train),len(X_test),len(X_cv))
```

```
1282192 356165 142466
```

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=5, max_features=50000)
train_vect1 = vectorizer.fit_transform(X_train)
test_vect1 = vectorizer.transform(X_test)
cv_vect1 = vectorizer.transform(X_cv)
```

In [109]:

```
print(train_vect1.shape)
print(test_vect1.shape)
print(cv_vect1.shape)
```

```
(1282192, 50000)
(356165, 50000)
(142466, 50000)
```

vectorizing test dataset

In [110]:

```
test_df = pd.read_csv("/content/drive/My Drive/Project/test.csv")
test_df.head()
```

Out[110]:

	id	comment_text
0	7097320	[Integrity means that you pay your debts.]\n\...
1	7097321	This is malfeasance by the Administrator and t...
2	7097322	@Rmiller101 - Spoken like a true elitist. But ...
3	7097323	Paul: Thank you for your kind words. I do, in...
4	7097324	Sorry you missed high school. Eisenhower sent ...

In [111]:

```
# Preprocessing the comments.
preprocessed_data_test = []
from tqdm import tqdm
for sent in tqdm(test_df['comment_text'].values):
    sent = process_sent(sent)
    line = ''
    for wrd in sent.split():
        if(len(wrd)>2):
            line += " " +wrd.lower()
    #line = re.sub(r"[']",'',line)
    preprocessed_data_test.append(line)

# Preparing the test data
test_X = vectorizer.transform(preprocessed_data)
```

```
100%|██████████| 97320/97320 [00:05<00:00, 16753.86it/s]
```

Training Logistic Regression on tfidf vectorization

On word unigrams

In [113]:

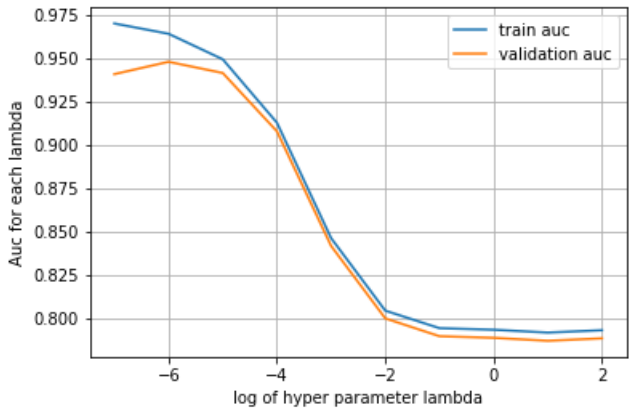
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
params = [0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,1.0,10,100]
train_auc = []
```

```
cv_auc = []
for i in tqdm(params):
    lr = SGDClassifier(loss='log',penalty='l2',tol=0.0001,alpha=i,n_jobs=-1,class_weight='balanced'
)
    model = CalibratedClassifierCV(lr,cv=5)
    model.fit(train_vect1,Y_train)
    predict_y_train = model.predict_proba(train_vect1)[:,1] # Taking probability for positive class
    t_auc = roc_auc_score(Y_train,predict_y_train)
    predict_y_cv = model.predict_proba(cv_vect1)[:,1]# Taking probability for positve class
    c_auc = roc_auc_score(Y_cv,predict_y_cv)
    train_auc.append(t_auc)
    cv_auc.append(c_auc)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
 10%|▉         | 1/10 [06:34<59:07, 394.17s/it]
 20%|██        | 2/10 [09:16<43:17, 324.68s/it]
 30%|███       | 3/10 [10:48<29:44, 254.91s/it]
 40%|████      | 4/10 [11:52<19:46, 197.69s/it]
 50%|█████     | 5/10 [12:39<12:41, 152.30s/it]
 60%|██████    | 6/10 [13:24<08:00, 120.14s/it]
 70%|███████   | 7/10 [13:57<04:42, 94.05s/it]
 80%|████████  | 8/10 [14:30<02:31, 75.80s/it]
 90%|█████████ | 9/10 [15:05<01:03, 63.53s/it]
100%|██████████| 10/10 [15:36<00:00, 53.68s/it]
```

In [114]:

```
import matplotlib.pyplot as plt
import math as m
para = [m.log10(x) for x in params]
plt.plot(para,train_auc,label='train auc')
plt.plot(para,cv_auc,label='validation auc')
plt.xlabel("log of hyper parameter lambda")
plt.ylabel("Auc for each lambda")
plt.grid()
plt.legend()
plt.show()
```



In [116]:

```
lr = SGDClassifier(loss='log',penalty='l2',tol=0.0001,alpha=0.000001,n_jobs=-1)
model = CalibratedClassifierCV(lr,cv=5)
model.fit(train_vect1,Y_train)
```

Out[116]:

```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=1e-06, average=False,
                                                    class_weight=None,
                                                    early_stopping=False,
                                                    epsilon=0.1, eta0=0.0,
                                                    fit_intercept=True,
                                                    l1_ratio=0.15,
                                                    learning_rate='optimal',
                                                    loss='log', max_iter=1000,
                                                    n_iter_no_change=5,
                                                    n_jobs=-1, penalty='l2',
                                                    power_t=0.5,
```

```
                                                    random_state=None,
                                                    shuffle=True, tol=0.0001,
                                                    validation_fraction=0.1,
                                                    verbose=0,
                                                    warm_start=False),
                        cv=5, method='sigmoid')
```

In [117]:

```
predict_y_test = model.predict_proba(test_vect1)[:,1] # Taking probability for positive class
eval = JigsawEvaluator(y_binary, y_identity_binary)
auc = eval.get_final_metric(predict_y_test)
print("Auc for logistic regression is {}".format(roc_auc_score(Y_test,predict_y_test)))
```

```
Auc for logistic regression is 0.9465540641624216
```

Training Naive bayes model

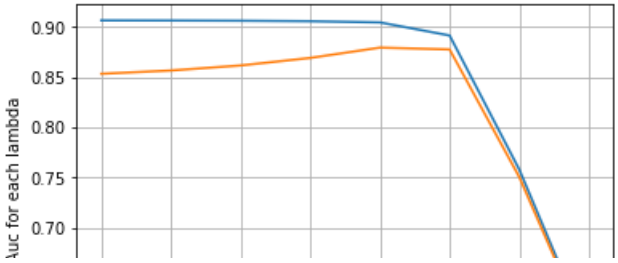
In [118]:

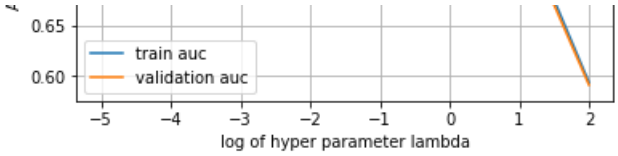
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

param = [0.00001,0.0001,0.001,0.01,0.1,1.0,10,100]
train_auc = []
cv_auc = []

for i in tqdm(param):
    model = MultinomialNB(alpha=i)
    model.fit(train_vect1, Y_train)
    predict_y_train = model.predict_log_proba(train_vect1)[:,1] # Taking probability for positive
class
    t_auc = roc_auc_score(Y_train,predict_y_train)
    predict_y_cv = model.predict_log_proba(cv_vect1)[:,1]# Taking probability for positve class
    c_auc = roc_auc_score(Y_cv,predict_y_cv)
    train_auc.append(t_auc)
    cv_auc.append(c_auc)
```

```
  0%|          | 0/8 [00:00<?, ?it/s]
 12%|█▎        | 1/8 [00:01<00:11,  1.61s/it]
 25%|██▌       | 2/8 [00:03<00:09,  1.61s/it]
 38%|███▊      | 3/8 [00:04<00:08,  1.61s/it]
 50%|█████     | 4/8 [00:06<00:06,  1.61s/it]
 62%|██████▎   | 5/8 [00:08<00:04,  1.61s/it]
 75%|███████▌  | 6/8 [00:09<00:03,  1.61s/it]
 88%|████████▊ | 7/8 [00:11<00:01,  1.60s/it]
100%|██████████| 8/8 [00:12<00:00,  1.60s/it]
```

In [119]:

```
import matplotlib.pyplot as plt
import math as m
para = [m.log10(x) for x in param]
plt.plot(para,train_auc,label='train auc')
plt.plot(para,cv_auc,label='validation auc')
plt.xlabel("log of hyper parameter lambda")
plt.ylabel("Auc for each lambda")
plt.grid()
plt.legend()
plt.show()
```





In [0]:

```
model = MultinomialNB(alpha=1)
model.fit(train_vect1, Y_train)
predict_y_test = model.predict_log_proba(test_vect1)[:,1] # Taking probability for positive class
print("Test auc for naive bayes model is {}".format(roc_auc_score(Y_test,predict_y_test)))
```

```
Test auc for naive bayes model is 0.8755791924694993
```

Using LSTM Models

In [0]:

```
#downloaded = drive.CreateFile({'id':id})
#glove_file = downloaded.GetContentFile('glove.6B.100d.txt')

f = open("/content/drive/My Drive/Project/glove.6B.300d.txt",'r',encoding="utf-8")
```

In [0]:

```
# Tokenizing the essay text data for train dataset
# We will train the vectorizer on train data and will use the same on test and cv data.
# Credit : https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
# Tokenizing the essay text data for train dataset
# We will train the vectorizer on train data and will use the same on test and cv data.
# Credit : https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense,Input
from numpy import asarray
t  = Tokenizer(num_words=40000)
t.fit_on_texts(train_x)
vocab_size = len(t.word_index)+1
# Integer coding all the words.
encoded_essay = t.texts_to_sequences(train_x)
# defining a max size for padding.
max_len = 150
# padding the vectors of each datapoint to fixed length of 600.
train_sequence = pad_sequences(encoded_essay,maxlen = max_len,padding='post')

# Vectorizing test data
# Integer coding all the words.
encoded_essay = t.texts_to_sequences(test_x)
# defining a max size for padding.
max_len = 150
# padding the vectors of each datapoint to fixed length of 600.
test_sequence = pad_sequences(encoded_essay,maxlen = max_len,padding='post')

# Vectorizing cv data
# Integer coding all the words.
encoded_essay = t.texts_to_sequences(cv_x)
# defining a max size for padding.
max_len = 150
# padding the vectors of each datapoint to fixed length of 600.
cv_sequence = pad_sequences(encoded_essay,maxlen = max_len,padding='post')

# we will load the whole glove vectors .
embeddings_index = {}
# Opening the file
f = open("/content/drive/My Drive/Project/glove.6B.300d.txt",'r',encoding="utf-8")
for line in f:
 values = line.split()
 word = values[0]
 coefs = asarray(values[1:], dtype='float32')
 embeddings_index[word] = coefs
```

```
f.close()
```

In [0]:

```
# Credit : https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
import numpy as np
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
 embedding_vector = embeddings_index.get(word)
 if embedding_vector is not None:
  embedding_matrix[i] = embedding_vector
```

In [28]:

```
#from google.colab import drive
#drive.mount('/content/drive')
test_df = pd.read_csv("/content/drive/My Drive/Project/test.csv")
test_df.head()
```

Out[28]:

	id	comment_text
0	7097320	[Integrity means that you pay your debts.]\n\...
1	7097321	This is malfeasance by the Administrator and t...
2	7097322	@Rmiller101 - Spoken like a true elitist. But ...
3	7097323	Paul: Thank you for your kind words. I do, in...
4	7097324	Sorry you missed high school. Eisenhower sent ...

In [29]:

```
# Preprocessing the comments.
test_preprocessed = []
bad_found = []
from tqdm import tqdm
for sent in tqdm(test_df['comment_text'].values):
    sent = process_sent(sent)
    line = ''
    for wrd in sent.split():
        if(len(wrd)>2):
            line += " " +wrd.lower()
    line = re.sub(r"[']",'',line)
    test_preprocessed.append(line)

test_df['comment_text'] = test_preprocessed
test_df.head()
```

```
100%|██████████| 97320/97320 [00:06<00:00, 15795.57it/s]
```

Out[29]:

	id	comment_text
0	7097320	integrity means that you pay your debts does ...
1	7097321	this malfeasance the administrator and the bo...
2	7097322	rmiller spoken like true elitist but look out...
3	7097323	paul thank you for your kind words indeed hav...
4	7097324	sorry you missed high school eisenhower sent ...

In [0]:

```
# Vectorizing test data
# Integer coding all the words.
encoded_essay1 = t.texts_to_sequences(test_df['comment_text'])
```

```
# defining a max size for padding.
max_len = 150
# padding the vectors of each datapoint to fixed length of 600.
test_sequence1 = pad_sequences(encoded_essay1,maxlen = max_len,padding='post')
```

First Architecture

In [66]:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import MaxPooling1D
from keras.layers import Flatten, GlobalMaxPooling1D, GlobalAveragePooling1D
from keras.layers import Dropout, BatchNormalization, Input, SpatialDropout1D
from keras.layers import Dense, Bidirectional, concatenate
from keras.models import Model
from keras.optimizers import Adam
from keras_self_attention import SeqSelfAttention
import keras.backend as K

embedding_vector_length = 100
input1 = Input(shape=(150,))
e1 = Embedding(40000,embedding_vector_length, input_length=150)(input1)
x1 = Bidirectional(LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.2))(e1)
att = SeqSelfAttention(attention_activation='sigmoid')(x1)

x1 = Flatten()(att)
output = Dense(1, activation='sigmoid')(x1)
model = Model(inputs=[input1,], outputs=[output])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy',])
model.summary()
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 150)               0
_________________________________________________________________
embedding_6 (Embedding)      (None, 150, 100)          4000000
_________________________________________________________________
bidirectional_6 (Bidirection (None, 150, 256)          234496
_________________________________________________________________
seq_self_attention_5 (SeqSel (None, 150, 256)          16449
_________________________________________________________________
flatten_5 (Flatten)          (None, 38400)             0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 38401
=================================================================
Total params: 4,289,346
Trainable params: 4,289,346
Non-trainable params: 0
_________________________________________________________________
```

In [67]:

```
model.fit(train_sequence, Y_train, nb_epoch=5, batch_size=512, validation_data=(cv_sequence, Y_cv))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: UserWarning: The `nb_epoch`
argument in `fit` has been renamed `epochs`.
  """Entry point for launching an IPython kernel.
```

```
Train on 1282192 samples, validate on 142466 samples
Epoch 1/5
1282192/1282192 [==============================] - 2665s 2ms/step - loss: 0.1483 - acc: 0.9454 - v
al_loss: 0.1334 - val_acc: 0.9491
Epoch 2/5
1282192/1282192 [==============================] - 2657s 2ms/step - loss: 0.1256 - acc: 0.9515 - v
al_loss: 0.1307 - val_acc: 0.9495
```

```
al_loss: 0.1507 - val_acc: 0.9499
Epoch 3/5
1282192/1282192 [==============================] - 2654s 2ms/step - loss: 0.1155 - acc: 0.9552 - v
al_loss: 0.1330 - val_acc: 0.9494
Epoch 4/5
1282192/1282192 [==============================] - 2659s 2ms/step - loss: 0.1037 - acc: 0.9600 - v
al_loss: 0.1407 - val_acc: 0.9477
Epoch 5/5
1282192/1282192 [==============================] - 2665s 2ms/step - loss: 0.0924 - acc: 0.9646 - v
al_loss: 0.1522 - val_acc: 0.9463
```

Out[67]:

```
<keras.callbacks.History at 0x7f0bd2deaf98>
```

In [69]:

```
# Roc auc
predicted_y = model.predict(test_sequence,batch_size=512)
from sklearn.metrics import roc_auc_score
print(roc_auc_score(Y_test,predicted_y))
```

```
0.9358755310733398
```

In [74]:

```
eval = JigsawEvaluator(y_binary, y_identity_binary)
print(eval.get_final_metric(predicted_y))
```

```
0.8823309859591929
```

In [0]:

```
y_pre = model.predict(test_sequence1,batch_size=512)
submission = pd.DataFrame({'id': test_df['id'].values})
submission['prediction'] = y_pre
submission.to_csv("submissionNew2.csv",index=False)
```

In [0]:

```
from google.colab import files
files.download( "/content/submissionNew2.csv" )
```

First Bi-LSTM architecture

In [0]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import MaxPooling1D
from keras.layers import Flatten, GlobalMaxPool1D
from keras.layers import Dropout, BatchNormalization
from keras.layers import Dense, Bidirectional
from keras.models import Model

embedding_vecor_length = 64
model = Sequential()
model.add(Embedding(vocab_size, embedding_vecor_length, input_shape=(150,)))
model.add(Bidirectional(LSTM(128)))
model.add(Dropout(0.3))
model.add(GlobalMaxPllo1D())
model.add(BatchNormalization())
model.add(Dense(768,activation='relu'))
model.add(Dropout(0.50))
model.add(GlobalMaxPllo1D())
model.add(BatchNormalization())
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_11 (Embedding)     (None, 200, 64)           6223104
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 256)               197632
_________________________________________________________________
dropout_15 (Dropout)         (None, 256)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dense_15 (Dense)             (None, 768)               197376
_________________________________________________________________
dropout_16 (Dropout)         (None, 768)               0
_________________________________________________________________
batch_normalization_6 (Batch (None, 768)               3072
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 769
=================================================================
Total params: 6,622,977
Trainable params: 6,620,929
Non-trainable params: 2,048
_________________________________________________________________
None
```

In [0]:

```
model.fit(train_sequence, Y_train, nb_epoch=5, batch_size=512, validation_data=(cv_sequence, Y_cv))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: UserWarning: The `nb_epoch`
argument in `fit` has been renamed `epochs`.
  """Entry point for launching an IPython kernel.
```

```
Train on 190911 samples, validate on 47728 samples
Epoch 1/5
190911/190911 [==============================] - 266s 1ms/step - loss: 0.2945 - acc: 0.8903 - val_
loss: 0.2760 - val_acc: 0.8859
Epoch 2/5
190911/190911 [==============================] - 259s 1ms/step - loss: 0.1441 - acc: 0.9477 - val_
loss: 0.1511 - val_acc: 0.9442
Epoch 3/5
190911/190911 [==============================] - 258s 1ms/step - loss: 0.1232 - acc: 0.9523 - val_
loss: 0.1696 - val_acc: 0.9411
Epoch 4/5
190911/190911 [==============================] - 257s 1ms/step - loss: 0.1061 - acc: 0.9584 - val_
loss: 0.1763 - val_acc: 0.9383
Epoch 5/5
190911/190911 [==============================] - 253s 1ms/step - loss: 0.0857 - acc: 0.9663 - val_
loss: 0.1948 - val_acc: 0.9394
```

Out[0]:

```
<keras.callbacks.History at 0x7f28de5322e8>
```

In [0]:

```
# Roc auc
predicted_y = model.predict(test_sequence,batch_size=512)
from sklearn.metrics import roc_auc_score
print(roc_auc_score(Y_test,predicted_y))
```

```
0.9038698075698687
```

Second Bi-LSTM architecture

In [0]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import MaxPooling1D
from keras.layers import Flatten, GlobalMaxPool1D
from keras.layers import Dropout, BatchNormalization, Input
from keras.layers import Dense, Bidirectional, concatenate
from keras.models import Model
from keras.optimizers import Adam

embedding_vector_length = 100
input1 = Input(shape=(150,))
x1 = Embedding(vocab_size,embedding_vector_length, input_length=150)(input1)
x1 = Bidirectional(LSTM(200,return_sequences=True))(x1)
x1 = Dropout(0.3)(x1)
x1 = GlobalMaxPool1D()(x1)

#input2 = Input(shape=(150,))
x2 = Embedding(vocab_size,embedding_vector_length, input_length=150)(input1)
x2 = Bidirectional(LSTM(200,return_sequences=True))(x2)
x2 = Dropout(0.5)(x2)
x2 = GlobalMaxPool1D()(x2)

x = concatenate([x1, x2])
x = Dense(75, activation='relu')(x)
x = Dropout(0.2)(x)
output = Dense(1, activation='sigmoid')(x)
model3 = Model(inputs=[input1,], outputs = output)
#adam = Adam(lr=0.0001)
model3.compile(loss='binary_crossentropy', optimizer= 'adam', metrics=['accuracy'])
model3.summary()
```

```
Model: "model_3"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_3 (InputLayer)         (None, 150)               0
_________________________________________________________________________________________________
embedding_5 (Embedding)      (None, 150, 100)          40000000    input_3[0][0]
_________________________________________________________________________________________________
embedding_6 (Embedding)      (None, 150, 100)          40000000    input_3[0][0]
_________________________________________________________________________________________________
bidirectional_5 (Bidirectional) (None, 150, 400)       481600      embedding_5[0][0]
_________________________________________________________________________________________________
bidirectional_6 (Bidirectional) (None, 150, 400)       481600      embedding_6[0][0]
_________________________________________________________________________________________________
dropout_7 (Dropout)          (None, 150, 400)          0           bidirectional_5[0][0]
_________________________________________________________________________________________________
dropout_8 (Dropout)          (None, 150, 400)          0           bidirectional_6[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_5 (GlobalM (None, 400)            0           dropout_7[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_6 (GlobalM (None, 400)            0           dropout_8[0][0]
_________________________________________________________________________________________________
concatenate_3 (Concatenate)  (None, 800)               0           global_max_pooling1d_5[0][0]
                                                                   global_max_pooling1d_6[0][0]
_________________________________________________________________________________________________
dense_5 (Dense)              (None, 75)                60075       concatenate_3[0][0]
_________________________________________________________________________________________________
dropout_9 (Dropout)          (None, 75)                0           dense_5[0][0]
_________________________________________________________________________________________________
dense_6 (Dense)              (None, 1)                 76          dropout_9[0][0]
=================================================================================================
Total params: 81,023,351
Trainable params: 81,023,351
Non-trainable params: 0
_________________________________________________________________________________________________
```

In [0]:

```
model3.fit(train_sequence, Y_train, epochs=4, batch_size=512, validation_data=[cv_sequence, Y_cv])
```

```
Train on 1282192 samples, validate on 142466 samples
```

```
Epoch 1/4
1282192/1282192 [==============================] - 4279s 3ms/step - loss: 0.1440 - acc: 0.9459 - v
al_loss: 0.1568 - val_acc: 0.9472
Epoch 2/4
1282192/1282192 [==============================] - 4267s 3ms/step - loss: 0.1159 - acc: 0.9536 - v
al_loss: 0.1421 - val_acc: 0.9489
Epoch 3/4
1282192/1282192 [==============================] - 4268s 3ms/step - loss: 0.0956 - acc: 0.9610 - v
al_loss: 0.1463 - val_acc: 0.9430
Epoch 4/4
1282192/1282192 [==============================] - 4270s 3ms/step - loss: 0.0711 - acc: 0.9714 - v
al_loss: 0.1569 - val_acc: 0.9380
```

Out[0]:

```
<keras.callbacks.History at 0x7f9dcd6d9b38>
```

In [0]:

```
# Roc auc
predicted_y = model3.predict(test_sequence,batch_size=512)
from sklearn.metrics import roc_auc_score
print(roc_auc_score(Y_test,predicted_y))
```

```
0.9373771542809413
```

In [0]:

```
y_pre = model3.predict(test_sequence1,batch_size=512)
submission = pd.DataFrame({'id': test_df['id'].values})
submission['prediction'] = y_pre
submission.to_csv("submission11.csv",index=False)
```

In [0]:

```
from google.colab import files
files.download( "/content/submission11.csv" )
```

Third Bi-LSTM architecture

In [0]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import MaxPooling1D
from keras.layers import Flatten, GlobalMaxPool1D
from keras.layers import Dropout, BatchNormalization, Input
from keras.layers import Dense, Bidirectional, concatenate
from keras.models import Model
from keras.optimizers import Adam

embedding_vector_length = 100
input1 = Input(shape=(150,))
x1 = Embedding(vocab_size,embedding_vector_length, input_length=150)(input1)
x1 = Bidirectional(LSTM(128,return_sequences=True))(x1)
x1 = GlobalMaxPool1D()(x1)


x2 = Embedding(vocab_size,embedding_vector_length, input_length=150)(input1)
x2 = Bidirectional(LSTM(128,return_sequences=True))(x2)
x2 = GlobalMaxPool1D()(x2)

x = concatenate([x1, x2])
x = Dense(64, activation='relu')(x)
x = Dropout(0.1)(x)
output = Dense(1, activation='sigmoid')(x)
model4 = Model(inputs=[input1,], outputs = output)
#adam = Adam(lr=0.0001)
```

```
model4.compile(loss='binary_crossentropy', optimizer= 'adam', metrics=['accuracy'])
model4.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0830 05:13:28.912689 139808314402688 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:66: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0830 05:13:28.966059 139808314402688 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:541: The name
tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

W0830 05:13:28.973107 139808314402688 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4432: The name
tf.random_uniform is deprecated. Please use tf.random.uniform instead.

W0830 05:13:30.026833 139808314402688 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:148: The name
tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default instea
d.

W0830 05:13:30.035221 139808314402688 deprecation.py:506] From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0830 05:13:30.064623 139808314402688 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name tf.train.Optimizer is dep
recated. Please use tf.compat.v1.train.Optimizer instead.

W0830 05:13:30.086747 139808314402688 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3657: The name tf.log i
s deprecated. Please use tf.math.log instead.

W0830 05:13:30.092208 139808314402688 deprecation.py:323] From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 150)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 150, 100)     40000000    input_1[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 150, 100)     40000000    input_1[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 150, 256)     234496      embedding_1[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 150, 256)     234496      embedding_2[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 256)          0           bidirectional_1[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_2 (GlobalM (None, 256)          0           bidirectional_2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 512)          0           global_max_pooling1d_1[0][0]
                                                                 global_max_pooling1d_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           32832       concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 64)           0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            65          dropout_1[0][0]
==================================================================================================
Total params: 80,501,889
Trainable params: 80,501,889
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [0]:

```
model4.fit(train_sequence, Y_train, epochs=4, batch_size=512, validation_data=[cv_sequence, Y_cv])
```

```
Train on 1282192 samples, validate on 142466 samples
Epoch 1/4
1282192/1282192 [==============================] - 3407s 3ms/step - loss: 0.1435 - acc: 0.9465 - v
al_loss: 0.1267 - val_acc: 0.9496
Epoch 2/4
1282192/1282192 [==============================] - 3402s 3ms/step - loss: 0.1150 - acc: 0.9541 - v
al_loss: 0.1284 - val_acc: 0.9495
Epoch 3/4
1282192/1282192 [==============================] - 3397s 3ms/step - loss: 0.0936 - acc: 0.9622 - v
al_loss: 0.1432 - val_acc: 0.9477
Epoch 4/4
1282192/1282192 [==============================] - 3387s 3ms/step - loss: 0.0662 - acc: 0.9737 - v
al_loss: 0.1730 - val_acc: 0.9422
```

Out[0]:

```
<keras.callbacks.History at 0x7f2704459c50>
```

In [0]:

```
# Train auc
predicted_y = model4.predict(train_sequence,batch_size=512)
from sklearn.metrics import roc_auc_score
print(roc_auc_score(Y_train,predicted_y))
```

```
0.9968805599682502
```

In [0]:

```
# Roc auc
predicted_y = model4.predict(test_sequence,batch_size=512)
from sklearn.metrics import roc_auc_score
print(roc_auc_score(Y_test,predicted_y))
```

```
0.937064717518582
```

In [0]:

```
y_pre = model4.predict(test_sequence1,batch_size=512)
submission = pd.DataFrame({'id': test_df['id'].values})
submission['prediction'] = y_pre
submission.to_csv("submission12.csv",index=False)
```

In [0]:

```
from google.colab import files
files.download( "/content/submission12.csv" )
```

Fourth Arcitecture

In [31]:

```
!pip install keras-self-attention
```

```
Collecting keras-self-attention
  Downloading
https://files.pythonhosted.org/packages/44/3e/eb1a7c7545eede073ceda2f5d78442b6cad33b5b750d7f074286
34b/keras-self-attention-0.42.0.tar.gz
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from keras-self-
attention) (1.16.5)
Requirement already satisfied: Keras in /usr/local/lib/python3.6/dist-packages (from keras-self-
attention) (2.2.5)
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/dist-packages (from Keras->
keras-self-attention) (1.3.1)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from Keras->keras-s
```

```
elf-attention) (2.8.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages (from Keras-
>keras-self-attention) (3.13)
Requirement already satisfied: keras-preprocessing>=1.1.0 in /usr/local/lib/python3.6/dist-
packages (from Keras->keras-self-attention) (1.1.0)
Requirement already satisfied: keras-applications>=1.0.8 in /usr/local/lib/python3.6/dist-packages
(from Keras->keras-self-attention) (1.0.8)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/dist-packages (from Keras-
>keras-self-attention) (1.12.0)
Building wheels for collected packages: keras-self-attention
  Building wheel for keras-self-attention (setup.py) ... done
  Created wheel for keras-self-attention: filename=keras_self_attention-0.42.0-cp36-none-any.whl s
ize=17296 sha256=1b6748e2be03d9d3f6e020ff3336076401e4edf7ec86551552a8fce209a7b832
  Stored in directory:
/root/.cache/pip/wheels/7b/05/a0/99c0cf60d383f0494e10eca2b238ea98faca9a1fe03cac2894
Successfully built keras-self-attention
Installing collected packages: keras-self-attention
Successfully installed keras-self-attention-0.42.0
```

In [50]:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import MaxPooling1D, Reshape
from keras.layers import Flatten, GlobalMaxPool1D, GlobalAveragePooling1D
from keras.layers import Dropout, BatchNormalization, Input, SpatialDropout1D
from keras.layers import Dense, Bidirectional, concatenate
from keras.models import Model
from keras.optimizers import Adam
from keras_self_attention import SeqSelfAttention


embedding_vector_length = 300
input1 = Input(shape=(150,))
e1 = Embedding(vocab_size,embedding_vector_length, input_length=150, weights=[embedding_matrix],
trainable=False)(input1)
x1 = Dropout(0.2)(e1)
x1 = Bidirectional(LSTM(120, return_sequences=True))(e1)
#lstm_att = SeqSelfAttention(attention_activation='sigmoid')(x1)

gru1, fh_state, bh_state = Bidirectional(GRU(60, return_sequences= True, return_state=True))(x1)
#x2 = SeqSelfAttention(attention_activation='sigmoid')(x2)

h_state = concatenate([fh_state, bh_state])
h_state = Reshape((-1,120))(h_state )

h_avg = GlobalAveragePooling1D()(gru1)
h_max = GlobalMaxPool1D()(gru1)

h_avg = Reshape((-1,120))(h_avg )
h_max = Reshape((-1,120))(h_max )


x = concatenate([h_state, h_avg, h_max])
x = Dense(20, activation='relu')(x)
x = Dropout(0.1)(x)
x = Flatten()(x)
#print(x)
output = Dense(1, activation='sigmoid')(x)
model5 = Model(inputs=[input1,], outputs = output)
#adam = Adam(lr=0.001)
model5.compile(loss= 'binary_crossentropy' , optimizer= 'adam', metrics=['accuracy',])
model5.summary()
```

```
Tensor("flatten_10/Reshape:0", shape=(?, ?), dtype=float32)
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_18 (InputLayer)        (None, 150)               0
_________________________________________________________________
embedding_18 (Embedding)     (None, 150, 300)          71461200
```

```
bidirectional_30 (Bidirectional (None, 150, 240)     404160      embedding_18[0][0]
bidirectional_31 (Bidirectional [(None, 150, 120), ( 108360      bidirectional_30[0][0]
concatenate_22 (Concatenate)    (None, 120)          0           bidirectional_31[0][1]
                                                                 bidirectional_31[0][2]
global_average_pooling1d_7 (Glo (None, 120)          0           bidirectional_31[0][0]
global_max_pooling1d_14 (Global (None, 120)          0           bidirectional_31[0][0]
reshape_12 (Reshape)            (None, 1, 120)       0           concatenate_22[0][0]
reshape_13 (Reshape)            (None, 1, 120)       0           global_average_pooling1d_7[0][0]
reshape_14 (Reshape)            (None, 1, 120)       0           global_max_pooling1d_14[0][0]
concatenate_23 (Concatenate)    (None, 1, 360)       0           reshape_12[0][0]
                                                                 reshape_13[0][0]
                                                                 reshape_14[0][0]
dense_7 (Dense)                 (None, 1, 20)        7220        concatenate_23[0][0]
dropout_21 (Dropout)            (None, 1, 20)        0           dense_7[0][0]
flatten_10 (Flatten)            (None, 20)           0           dropout_21[0][0]
dense_8 (Dense)                 (None, 1)            21          flatten_10[0][0]
==================================================================================================
Total params: 71,980,961
Trainable params: 519,761
Non-trainable params: 71,461,200
__________________________________________________________________________________________________
```

In [51]:

```
model5.fit(train_sequence, Y_train, epochs=6, batch_size=512, validation_data=[cv_sequence, Y_cv])
```

```
Train on 1282192 samples, validate on 142466 samples
Epoch 1/6
1282192/1282192 [==============================] - 2724s 2ms/step - loss: 0.1427 - acc: 0.9465 - v
al_loss: 0.1264 - val_acc: 0.9503
Epoch 2/6
1282192/1282192 [==============================] - 2730s 2ms/step - loss: 0.1237 - acc: 0.9516 - v
al_loss: 0.1253 - val_acc: 0.9507
Epoch 3/6
1282192/1282192 [==============================] - 2724s 2ms/step - loss: 0.1171 - acc: 0.9535 - v
al_loss: 0.1219 - val_acc: 0.9519
Epoch 4/6
1282192/1282192 [==============================] - 2713s 2ms/step - loss: 0.1104 - acc: 0.9556 - v
al_loss: 0.1253 - val_acc: 0.9505
Epoch 5/6
1282192/1282192 [==============================] - 2713s 2ms/step - loss: 0.1026 - acc: 0.9581 - v
al_loss: 0.1263 - val_acc: 0.9506
Epoch 6/6
1282192/1282192 [==============================] - 2724s 2ms/step - loss: 0.0933 - acc: 0.9612 - v
al_loss: 0.1340 - val_acc: 0.9492
```

Out[51]:

```
<keras.callbacks.History at 0x7f0cc1757d68>
```

In [0]:

```
# Creating loss metric
j_eval = JigsawEvaluator(y_binary, y_identity_binary)
```

In [0]:

```
predicted_y = model5.predict(test_sequence,batch_size=512)
final_auc = j_eval.get_final_metric(predicted_y)
```

In [54]:

```
# This is the auc metric that was given by kaggle.
print("Auc of the model is {}".format(final_auc))
```

```
Auc of the model is 0.9118993070137469
```

In [0]:

```
predicted_y1 = (predicted_y >= 0.5).astype(int)
```

In [56]:

```
# This is the simple auc on test data
print(roc_auc_score(Y_test, predicted_y))
```

```
0.9550143824173905
```

In [0]:

```
predict_y = model5.predict(test_sequence1,batch_size=512)
```

In [0]:

```
# saving model to google drive
from keras.models import load_model
model5.save("/content/drive/My Drive/Project/my_model5.h5")
```

In [0]:

```
#y_pre = model5.predict(test_sequence1,batch_size=32)
submission = pd.DataFrame({'id': test_df['id'].values})
submission['prediction'] = predict_y
submission.to_csv("submission18.csv",index=False)
```

In [0]:

```
from google.colab import files
files.download( "/content/submission18.csv" )
```

Conclusion

In [3]:

```
from prettytable import PrettyTable

x = PrettyTable(['model','vectorizer','test auc'])
x.add_row(['Logistic Regression','tfidf unigram','0.9465'])
x.add_row(['Naive Bayes','tfidf unigram','0.8755'])
x.add_row(['1 layered Bi-LSTM + self attention layer','glove 100dim ','0.8823'])
x.add_row(['1 layered Bi-LSTM','glove 100dim','0.9038'])
x.add_row(['2 layered Bi-LSTM(200 units)','glove 100dim ','0.9373'])
x.add_row(['2 layered Bi-LSTM(128 units)','glove 100dim ','0.9370'])
x.add_row(['1 layered Bi-LSTM(120 units) + 1 layered Bi-gru(60) units','glove 300dim ','0.9550'])
print(x)
```

```
+-----------------------------------------------------------+---------------+----------+
|                           model                           |   vectorizer  | test auc |
+-----------------------------------------------------------+---------------+----------+
|                    Logistic Regression                    | tfidf unigram |  0.9465  |
|                        Naive Bayes                        | tfidf unigram |  0.8755  |
|          1 layered Bi-LSTM + self attention layer         | glove 100dim  |  0.8823  |
|                     1 layered Bi-LSTM                     |  glove 100dim |  0.9038  |
|                2 layered Bi-LSTM(200 units)               | glove 100dim  |  0.9373  |
|                2 layered Bi-LSTM(128 units)               | glove 100dim  |  0.9370  |
| 1 layered Bi-LSTM(120 units) + 1 layered Bi-gru(60) units | glove 300dim  |  0.9550  |
```

| 1 layered Bi LSTM(120 units) + 1 layered Bi GRU(60 units) | glove 300dim | 0.9550 |
+--+---------------+----------+

The best model that we had was the fourth and the last model that gave us an auc of 0.9176 on the kaggle private leader board.