In [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter


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

# Credits: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networ
ks-python-keras/
# LSTM for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
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
# fix random seed for reproducibility
numpy.random.seed(7)
from numpy import array
from numpy import asarray
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten

from keras.layers import Embedding
from keras.layers import Dense,LSTM
from keras import Input
import numpy as np
np.random.seed(0)
from keras.models import Model
from keras.layers import Dense, Input, Dropout, LSTM, Activation,Reshape
```

```
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.initializers import glorot_uniform
import keras
from keras.models import load_model, Model
from keras.layers import Dense, Activation, Dropout, Input, LSTM, Reshape, Lambda, RepeatVector
from keras.initializers import glorot_uniform
from keras.utils import to_categorical
from keras.optimizers import Adam
from keras import backend as K
from sklearn.metrics import roc_auc_score
from sklearn.datasets import make_classification
from keras.models import Sequential
import tensorflow as tf
from sklearn.metrics import roc_auc_score

from keras.layers import Dense
from keras.utils import np_utils
from keras.callbacks import Callback, EarlyStopping
```

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

```
Using TensorFlow backend.
```

In [0]:

```
dataset = pd.read_pickle('gdrive/My Drive/dataset_naivebayes_original.pkl')
```

In [0]:

```
dataset = dataset[:69999]
```

In [0]:

```
dataset.shape
```

Out[0]:

```
(69999, 122)
```

We are trying to pick the random rows from Dataset instead of sequential Rows
From the index we try to pick the random rows for train,cv,test
After randomising Dataset,[:45000] are Train,[45000:60000] are CV , [60000:] are test.

In [0]:

```
index=np.random.choice(np.arange(len(dataset)),size=69999)
```

In [0]:

```
index
```

Out[0]:

```
array([68268, 43567, 42613, ..., 40721,  5358,  1324])
```

In [0]:

```
len(index)
```

Out[0]:

```
69999
```

In [0]:

```
dataset  = dataset.iloc[index,:]
```

In [0]:

```
dataset.head()
```

Out[0]:

	AK	AL	AR	AZ	CA	CO	CT	DC	DE	FL	GA	HI	IA	ID	IL	IN	KS	KY	LA	MA	MD	ME	MI	MN	MO	MS	MT
68268	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
43567	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
42613	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45891	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21243	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 122 columns

In [0]:

```
index_list = dataset.index.tolist()
```

In [0]:

```
dataset = dataset.reset_index(drop=True)
```

In [0]:

```
dataset.head()
```

Out[0]:

	AK	AL	AR	AZ	CA	CO	CT	DC	DE	FL	GA	HI	IA	ID	IL	IN	KS	KY	LA	MA	MD	ME	MI	MN	MO	MS	MT	NC
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 122 columns

In [0]:

```
preprocessed_essays = dataset['preprocessed_essays']
```

In [0]:

```
from sklearn.preprocessing import OneHotEncoder
onehot_encoder = OneHotEncoder()

y_encoded = onehot_encoder.fit_transform(dataset[['y']]).toarray()
print('outputs_encoded.shape after One Hot Encode', y_encoded.shape)
```

```
outputs_encoded.shape after One Hot Encode (69999, 2)
```

Model_1

Intuition behind Train,test,cv for text-data

We try to use the words present in the train data[:45000] and preprocess TEST,CV such that test-text and cv-test contain only words which are present in the train..(which is similar to the fit_transform)

In [0]:

```
vectorizer3 = TfidfVectorizer(min_df=5)
essay_processed = vectorizer3.fit_transform(preprocessed_essays[:45000])
```

In [0]:

```
words_df  = pd.DataFrame([vectorizer3.idf_,vectorizer3.vocabulary_]).T

words_df.head()
```

Out[0]:

	0	1
0	7.35773	it
1	5.868	time
2	9.76853	put
3	9.92268	stem
4	9.92268	as

In [0]:

```
words_tfidf_list = list(words_df[1])
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
```

```
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(preprocessed_essays[45000:60000].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e in words_tfidf_list)
    preprocessed_essays_cv.append(sent.lower().strip())
```

```
100%|██████████| 15000/15000 [01:00<00:00, 247.10it/s]
```

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentance in tqdm(preprocessed_essays[60000:].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e in words_tfidf_list)
    preprocessed_essays_test.append(sent.lower().strip())
```

```
100%|██████████| 9999/9999 [00:40<00:00, 245.30it/s]
```

Intuition behind the Train,Test,cv for PADDED_DOCS

the input of text is 3-dimensional
Each text do not have more that 316 words so we pad them with max_length-==350 words because all text-sentence must have same words

padded_docs do store the index(rank) of words for a particular text...
We will be passing this through the embedding layer which gives output of each word as vector..
input == padded_docs(index of words) ----to----embedding Layer -----output == Word_Vector of each vector based on index

Tokenizer should be trained with train_Set([:45000]) and the index for each word should be noted

Then the fitted tokenizer should be used top Transform the CV and TEST-DATA into index's.

Input_Text

In [0]:

```
maxLen = len(max(preprocessed_essays[:45000], key=len).split())
maxLen
```

Out[0]:

```
316
```

In [0]:

In [0]:

```
from keras.preprocessing.text import Tokenizer

t = Tokenizer()
t.fit_on_texts(preprocessed_essays[:45000])
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(preprocessed_essays[:45000])
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 350
padded_docs = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
[[ 54  35 375 ...   0   0   0]
 [  4 811   1 ...   0   0   0]
 [  2  47 290 ...   0   0   0]
 ...
 [ 28  30   3 ...   0   0   0]
 [  2  47 746 ...   0   0   0]
 [ 24   6 637 ...   0   0   0]]
```

In [0]:

```
from keras.preprocessing.text import Tokenizer



# integer encode the documents
encoded_docs = t.texts_to_sequences(preprocessed_essays_cv)
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 350
padded_docs_cv = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs_cv)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
[[    4     1  3543 ...     0     0     0]
 [    4     1    94 ...     0     0     0]
 [    4     1    30 ...     0     0     0]
 ...
 [   47   568     3 ...     0     0     0]
 [   78    89   135 ...     0     0     0]
 [10723  8552 11600 ...     0     0     0]]
```

In [0]:

```
from keras.preprocessing.text import Tokenizer



# integer encode the documents
encoded_docs = t.texts_to_sequences(preprocessed_essays_test)
```

```
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 350
padded_docs_test = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs_test)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [0]:

```
len(padded_docs)
```

Out[0]:

```
45000
```

In [0]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('gdrive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

Importing Glove_vector and we form embedding_matrix such that each word has a vector
This embedding matrix behave as the weights of embedding Layer

In [0]:

```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))
embedding_matrix.shape
```

Out[0]:

```
(35470, 300)
```

In [0]:

```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
  if word in glove_words and word in t.word_index.keys():
    embedding_vector = model[word]
    if embedding_vector is not None:

      embedding_matrix[i] = embedding_vector
```

Text-Input-Layer

the output from the embedding later is passed to the LSTM

In [0]:

```
sequence_input = Input(shape=(max_length,), dtype='int32')
embedded_sequences = Embedding(vocab_size,output_dim=300,weights=[embedding_matrix],input_length=350,trainable=False)(sequence_input)
lstm_out = LSTM(64,return_sequences=True)(embedded_sequences)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0721 12:04:59.609628 140648298743680 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:74: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0721 12:04:59.630064 140648298743680 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:517: The name
tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

W0721 12:04:59.633968 140648298743680 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4138: The name
tf.random_uniform is deprecated. Please use tf.random.uniform instead.

W0721 12:04:59.650957 140648298743680 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:174: The name
tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_session instead.

W0721 12:04:59.651927 140648298743680 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:181: The name
tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.
```

The output of LSTM is Flattened out,this means the three dimensional output of embedding layer is converted into Two-Dimensional

In [0]:

```
flatten_layer = Flatten()
X = flatten_layer(lstm_out)
```

School State

School_state-Input

In [0]:

```
project_data = pd.read_csv('gdrive/My Drive/train_data.csv')
```

In [0]:

```
projet_data = project_data[:69999]
```

In [0]:

```
project_data.head()
```

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

project_data should also have the similar index's to the dataset and So, we use index of dataset in project_data

In [0]:

```
project_data  = project_data.iloc[index,:]
```

In [0]:

```
project_data.head()
```

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
68268	18586	p033319	6c5a2dc45d07f197659476dd2ade234b	Mr.	CA	2016-08-31 13:29:25
43567	150015	p080320	96aa7607d0bffc29749156e9242db5d8	Mrs.	CA	2016-08-11 01:17:03
42613	113082	p028568	68645a7f4afb26f30471c653cb897fe8	Ms.	MA	2017-04-23 20:19:58
45891	64337	p240233	fb1efc328e16ff24e025c0dce0e5c26c	Mrs.	CA	2016-12-29 12:42:51
21243	158345	p121156	c9df8d14efc4aa0f89c9a94e05667273	Mrs.	NC	2017-03-04 21:39:08

In [0]:

```
project_data = project_data.reset_index(drop=True)
```

In [0]:

```
project_data.head()
```

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	18586	p033319	6c5a2dc45d07f197659476dd2ade234b	Mr.	CA	2016-08-31 13:29:25	Gra

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
1	150015	p080320	96aa7607d0bffc29749156e9242db5d8	Mrs.	CA	2016-08-11 01:17:03	Gra
2	113082	p028568	68645a7f4afb26f30471c653cb897fe8	Ms.	MA	2017-04-23 20:19:58	Gra
3	64337	p240233	fb1efc328e16ff24e025c0dce0e5c26c	Mrs.	CA	2016-12-29 12:42:51	Gra
4	158345	p121156	c9df8d14efc4aa0f89c9a94e05667273	Mrs.	NC	2017-03-04 21:39:08	Gra

In [0]:

```
school_state_input = project_data['school_state'][:69999]
len(set(school_state_input))
```

Out[0]:

```
51
```

This maybe similar to the text but Text has many time steps here and Categorical_Variables have only one Time-Step.
We could even Pad the Cat_variables(if needed we can have max_length =10 which means input is padded)
The padded_docs_school stores the index of cat_variables based on rank (which occurs too many times)
Input is passed to Text and output of emedding gives the Embedding_vectors for each Cat_variable

In [0]:

```
from keras.preprocessing.text import Tokenizer

t = Tokenizer()
t.fit_on_texts(school_state_input[:69999])
vocab_size = len(t.word_index) +1
encoded_docs = t.texts_to_sequences(school_state_input)

# integer encode the documents


# pad documents to a max length of 4 words
max_length = 1
padded_docs_school = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
```

In [0]:

```
vocab_size
```

Out[0]:

```
52
```

Input_layer_School_state

input length =1,because as we have not padded the cat_variable
input is passed to embedding layer and the output is embedded vector for each cat_variable
The output layer is passed to Flatten_layer

In [0]:

In [0]:

```
input_layer_school = Input(shape=(1,))
embedding_school = Embedding(51+1,output_dim= 3, input_length=1)(input_layer_school)
flatten_school = Flatten()(embedding_school)
```

Grade

Input_Grade

The problem here is we could use the earlier methods but it splits a single cat_variable into multiple cat_variables

So,we need to store the index(rank) based on value_counts() of the grade in the padded_docs_grade and below is the code for this method

We need to pass this padded_docs_grade into the embedding_layer

And the rest is similar to the school_state process

In [0]:

```
z = project_data['project_grade_category'][:69999].value_counts()
z_index = list(z.index)

indices = []

for j in range(0,len(z_index)):
  index_list = []
  for i in range(0,69999):

    if(project_data['project_grade_category'][i] == z_index[j]):
      index_list.append(i)
  indices.append(index_list)


for i in range(0,len(z_index)):

  project_data['project_grade_category'][indices[i]] = i+1
```

In [0]:

```
grade_category_input = project_data['project_grade_category'][:69999]
grade_category_input = [[i] for i in list(grade_category_input)]
```

input_Grade_Layer

In [0]:

```
input_layer_grade = Input(shape=(1,))
embedding_grade = Embedding(input_dim=4+1,output_dim= 2, input_length=1)(input_layer_grade)
flatten_grade = Flatten()(embedding_grade)
```

In [0]:

```
padded_docs_grade = np.array(grade_category_input)
```

Clean_categories

Input_Clean_Categories

In [0]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
```

```
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())
```

In [0]:

```
project_data['clean_categories'] = cat_list
```

In [0]:

```
project_data['clean_categories'].nunique()
```

Out[0]:

```
49
```

In [0]:

```
z = project_data['clean_categories'][:69999].value_counts()
z_index = list(z.index)

indices = []

for j in range(0,len(z_index)):
  index_list = []
  for i in range(0,69999):

    if(project_data['clean_categories'][i] == z_index[j]):
      index_list.append(i)
  indices.append(index_list)


for i in range(0,len(z_index)):

  project_data['clean_categories'][indices[i]] = i+1
```

In [0]:

```
clean_categories_input = project_data['clean_categories'][:69999]
clean_categories_input = [[i] for i in list(clean_categories_input)]
```

Input_Clean_Categories_Layer

In [0]:

```
input_layer_categories = Input(shape=(1,))
embedding_categories = Embedding(input_dim=51+1,output_dim= 3, input_length=1)
(input_layer_categories)
flatten_categories = Flatten()(embedding_categories)
```

In [0]:

```
padded_docs_categories = np.array(clean_categories_input)
```

```
padded_docs_categories = np.array(clean_categories_input)
```

Clean_SubCategories

Input_clean_SubCategories

In [0]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
```

In [0]:

```
project_data['clean_subcategories'].nunique()
```

Out[0]:

```
374
```

In [0]:

```
z = project_data['clean_subcategories'][:69999].value_counts()
z_index = list(z.index)

indices = []

for j in range(0,len(z_index)):
  index_list = []
  for i in range(0,69999):

    if(project_data['clean_subcategories'][i] == z_index[j]):
      index_list.append(i)
  indices.append(index_list)


for i in range(0,len(z_index)):

  project_data['clean_subcategories'][indices[i]] = i+1
```

In [0]:

```
clean_subcategories_input = project_data['clean_subcategories'][:69999]
clean_subcategories_input = [[i] for i in list(clean_subcategories_input)]
```

Input_Clean_SubCategories_Layer

In [0]:

```
input_layer_subcategories = Input(shape=(1,))
embedding_subcategories = Embedding(input_dim=401+1,output_dim= 10, input_length=1)(input_layer_sub
categories)
flatten_subcategories = Flatten()(embedding_subcategories)
```

In [0]:

```
padded_docs_subcategories = np.array(clean_subcategories_input)
```

Teacher_prefix

Input_prefix

In [0]:

```
project_data['teacher_prefix'].nunique()
```

Out[0]:

```
5
```

In [0]:

```
#replacing nan values in pandas    https://stackoverflow.com/questions/13295735/how-can-i-replace-
all-the-nan-values-with-zeros-in-a-column-of-a-pandas-datafra
project_data['teacher_prefix'].value_counts()
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'].isnull().any()
```

Out[0]:

```
False
```

In [0]:

```
teacher_prefix_input = project_data['teacher_prefix'][:69999]
```

In [0]:

```
from keras.preprocessing.text import Tokenizer

t = Tokenizer()
t.fit_on_texts(teacher_prefix_input[:69999])
vocab_size = len(t.word_index)
encoded_docs = t.texts_to_sequences(teacher_prefix_input)


# integer encode the documents


# pad documents to a max length of 4 words
max_length = 1
padded_docs_prefix = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
```

Input_Prefix_Layer

In [0]:

```
input_layer_prefix = Input(shape=(1,))
embedding_prefix = Embedding(input_dim=5+1,output_dim= 2, input_length=1)(input_layer_prefix)
flatten_prefix = Flatten()(embedding_prefix)
```

Input_Numerical_Features

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

min_max_scaler_price = preprocessing.MinMaxScaler()
price_data = min_max_scaler_price.fit_transform(dataset[['price_standardized']])
```

In [0]:

```
min_max_scaler_quantity = preprocessing.MinMaxScaler()
quantity_data = min_max_scaler_quantity.fit_transform(dataset[['quantity_standardised']])
```

In [0]:

```
min_max_scaler_digits = preprocessing.MinMaxScaler()
digits_data = min_max_scaler_digits.fit_transform(dataset[['digits_standardised']])
```

In [0]:

```
min_max_scaler_previous = preprocessing.MinMaxScaler()
previous_project_data =
min_max_scaler_previous.fit_transform(dataset[['previously_posted_projects']])
```

Input_Numerical_Layers

In [0]:

```
previous_project = Input(shape=(1,), dtype='float32')
digits = Input(shape=(1,), dtype='float32')
price = Input(shape=(1,),dtype='float32')
quantity = Input(shape=(1,),dtype='float32')


numerical_concatenate = keras.layers.concatenate([previous_project,digits,price,quantity],axis=-1)
numerical_dense = Dense(1, activation='relu')(numerical_concatenate)
```

Concatenate all the Text,Categorical and Numerical Layers-outputs

In [0]:

```
e_concatenate = keras.layers.concatenate([X,
                                          flatten_school,
                                          flatten_grade,
                                          flatten_categories,
                                          flatten_subcategories,
                                          flatten_prefix,
                                          numerical_dense],axis=-1)


Dense_1 = Dense(32, activation='relu')(e_concatenate)
dropout_1 = Dropout(0.25)(Dense_1)
Dense_2 = Dense(16, activation='relu')(dropout_1)
dropout_2 = Dropout(0.5)(Dense_2)
Dense_3 = Dense(8, activation='relu')(dropout_2)
predictions = Dense(2, activation='sigmoid')(Dense_3)
```

Input for all Numerical,Categorical,Text Features(Layer-Inputs)

In [0]:

```
input_all = []
input_all = [sequence_input,
             input_layer_school,
```

```
            input_layer_grade,
            input_layer_categories,
            input_layer_subcategories,
            input_layer_prefix,
            previous_project,
            digits,
            price,
            quantity]
```

Inputs and output for the Model

In [0]:

```
model_1 = Model(inputs= input_all, outputs = predictions)
```

In [0]:

```
model_1.summary()
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 350)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 350, 300)     10641000    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_3 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_4 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_5 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_6 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_7 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_8 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_9 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_10 (InputLayer)           (None, 1)            0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 350, 64)      93440       embedding_1[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 3)         156         input_2[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 2)         10          input_3[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 3)         156         input_4[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 10)        4020        input_5[0][0]
__________________________________________________________________________________________________
embedding_6 (Embedding)         (None, 1, 2)         12          input_6[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 4)            0           input_7[0][0]
                                                                 input_8[0][0]
                                                                 input_9[0][0]
                                                                 input_10[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 22400)        0           lstm_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 3)            0           embedding_2[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 2)            0           embedding_3[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten)             (None, 3)            0           embedding_4[0][0]
__________________________________________________________________________________________________
flatten_5 (Flatten)             (None, 10)           0           embedding_5[0][0]
__________________________________________________________________________________________________
flatten_6 (Flatten)             (None, 2)            0           embedding_6[0][0]
__________________________________________________________________________________________________
```

```
dense_1 (Dense)                 (None, 1)            5           concatenate_1[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 22421)        0           flatten_1[0][0]
                                                                 flatten_2[0][0]
                                                                 flatten_3[0][0]
                                                                 flatten_4[0][0]
                                                                 flatten_5[0][0]
                                                                 flatten_6[0][0]
                                                                 dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 32)           717504      concatenate_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 32)           0           dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 16)           528         dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 16)           0           dense_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 8)            136         dropout_2[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 2)            18          dense_4[0][0]
==================================================================================================
Total params: 11,456,985
Trainable params: 815,985
Non-trainable params: 10,641,000
__________________________________________________________________________________________________
```

In [0]:

```
import tensorflow as tf
from sklearn.metrics import roc_auc_score

def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
opt = Adam(lr=0.01, beta_1=0.9, beta_2=0.999, decay=0.01)
model_1.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy',auroc])
```

In [0]:

```
model_1.fit([padded_docs[:45000],
           padded_docs_school[:45000],
           padded_docs_grade[:45000],
           padded_docs_categories[:45000],
           padded_docs_subcategories[:45000],
           padded_docs_prefix[:45000],
           previous_project_data[:45000],
           digits_data[:45000],
           price_data[:45000],
           quantity_data[:45000]],y_encoded[:45000],validation_data=([padded_docs_cv,
           padded_docs_school[45000:60000],
           padded_docs_grade[45000:60000],
           padded_docs_categories[45000:60000],
           padded_docs_subcategories[45000:60000],
           padded_docs_prefix[45000:60000],
           previous_project_data[45000:60000],
           digits_data[45000:60000],
           price_data[45000:60000],
           quantity_data[45000:60000]],y_encoded[45000:60000]),epochs=4,batch_size=512)
```

```
Train on 45000 samples, validate on 15000 samples
Epoch 1/4
45000/45000 [==============================] - 308s 7ms/step - loss: 0.4534 - acc: 0.8338 - auroc:
0.5773 - val_loss: 0.4130 - val_acc: 0.8442 - val_auroc: 0.6996
Epoch 2/4
45000/45000 [==============================] - 304s 7ms/step - loss: 0.3992 - acc: 0.8490 - auroc:
0.6907 - val_loss: 0.3917 - val_acc: 0.8449 - val_auroc: 0.7247
Epoch 3/4
45000/45000 [==============================] - 304s 7ms/step - loss: 0.3799 - acc: 0.8530 - auroc:
0.7304 - val_loss: 0.3993 - val_acc: 0.8357 - val_auroc: 0.7314
Epoch 4/4
```

```
45000/45000 [==============================] - 307s 7ms/step - loss: 0.3647 - acc: 0.8559 - auroc:
0.7516 - val_loss: 0.3778 - val_acc: 0.8491 - val_auroc: 0.7471
```

Out[0]:

```
<keras.callbacks.History at 0x7f8372ed5860>
```

In [0]:

```
y_predicted = model_1.predict([padded_docs_test,
               padded_docs_school[60000:],
               padded_docs_grade[60000:],
               padded_docs_categories[60000:],
               padded_docs_subcategories[60000:],
               padded_docs_prefix[60000:],
               previous_project_data[60000:],
               digits_data[60000:],
               price_data[60000:],
               quantity_data[60000:]])
```

In [0]:

```
test_auc = roc_auc_score(y_encoded[60000:], y_predicted)
print("test_auc",test_auc)
```

```
test_auc 0.74918674961027
```

In [0]:

```
from keras import backend as K

K.clear_session()
```

Model-2

In [0]:

```
dataset = pd.read_pickle('gdrive/My Drive/dataset_naivebayes_original.pkl')
```

In [0]:

```
dataset = dataset[:69999]
```

In [0]:

```
dataset  = dataset.iloc[index,:]
```

In [0]:

```
dataset = dataset.reset_index(drop=True)
```

In [0]:

```
preprocessed_essays = dataset['preprocessed_essays'][:45000]
```

Tfidf-Vectorizer

In [0]:

```
vectorizer3 = TfidfVectorizer(min_df=5)
essay_processed = vectorizer3.fit_transform(preprocessed_essays)
```

In [0]:

```
print(vectorizer3.idf_)
```

```
[7.35773116 5.86800122 9.76852984 ... 9.22953334 9.92268052 8.71870772]
```

Thses are the IDF Values

In [0]:

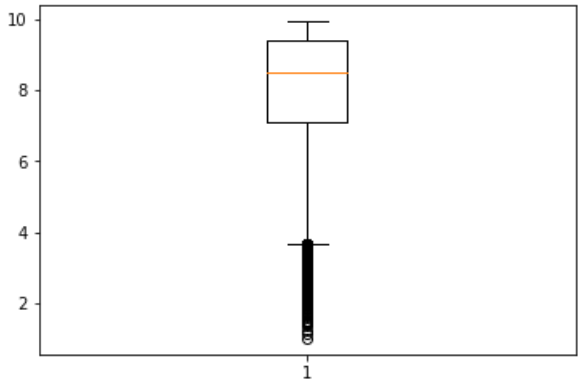
```
print(len(vectorizer3.vocabulary_))
```

```
16063
```

these are the total no.of.words

In [0]:

```
plt.boxplot(vectorizer3.idf_)
plt.show()
```



This is the Box-Plot for the idf_ values

we need to remove the values which are more frequent and Less-Frequent

In [0]:

```
print(np.mean(vectorizer3.idf_))

print(np.percentile(vectorizer3.idf_,np.arange(0,100,5)))
```

```
8.04881496915592
[1.0075169  4.66171894 5.55788518 6.20222337 6.71049368 7.1092698
 7.46594475 7.76319627 8.00086792 8.24870409 8.49556417 8.71870772
 8.88122665 9.07538266 9.22953334 9.4118549  9.51721541 9.63499845
 9.76852984 9.92268052]
```

In [0]:

```
words_df  = pd.DataFrame([vectorizer3.idf_,vectorizer3.vocabulary_]).T

words_df.head()
```

Out[0]:

	0	1
0	7.35773	it
1	5.868	time
2	9.76853	put
3	9.92268	stem

4	9.92268	as

In [0]:

```
words_tfidf  = words_df[words_df[0]<=9.85366543][words_df[0]>= 6.98576653]

words_tfidf_list = words_tfidf[1]

words_tfidf_list = list(words_tfidf_list)
```

We are setting certain threshold's for idf_values and removing more frequent and less-Frequent words.
Store all those words in the list

In [0]:

```
len(words_tfidf_list)
```

Out[0]:

```
11048
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_list = []
# tqdm is for printing the status bar
for sentance in tqdm(preprocessed_essays.values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e in words_tfidf_list)
    preprocessed_essays_list.append(sent.lower().strip())
```

```
100%|██████████| 45000/45000 [18:38<00:00, 40.23it/s]
```

Now from the Preprocessed_essays, check the words in the list and Preprocess the essays

store the preprocessed data in the preprocessed_essays_list which has only words in the threshold limits

In [0]:

```
print(len(preprocessed_essays[1]))
preprocessed_essays[1]
```

```
773
```

Out[0]:

```
'my beautiful students five year old little boys girls latino cultural background they live neighb
orhoods mid city los angeles go title one community school they 100 english language learners eage
r learn fun most students not school experience entering classroom they generally shocked school e
xperience assimilate grow i proud much students accomplish year my students starting educational j
ourneys as lot people might know kindergarteners age routine organization important therefore plac
e store belongings utmost importance this cubbie idea much needed allows consistent organization a
lso calls responsibility students they need learn put belongings know get materials furthermore de
signated time place organize beneficial life this way everything organized ready purpose'
```

In [0]:

```
print(len(preprocessed_essays_list[1]))
preprocessed_essays_list[1]
```

```
490
```

Out[0]:

```
'my beautiful five old little girls cultural background they live city los school they 100 english
learners learn fun most not school entering classroom they generally school assimilate much my jou
rneys people might know kindergarteners routine organization important therefore store belongings
utmost importance this cubbie idea much needed allows consistent organization also calls they need
learn put belongings know materials furthermore designated organize this everything ready purpose'
```

Check the values of the preprocesed_essays_list and preprocesed_essays

We can find that the most frequent words and least frequent words are missing from preprocesed_essays_list

In [0]:

```
vectorizer3 = TfidfVectorizer(min_df=5)
essay_processed = vectorizer3.fit_transform(preprocessed_essays_list[:45000])

words_df  = pd.DataFrame([vectorizer3.idf_,vectorizer3.vocabulary_]).T

words_df.head()

words_tfidf_list = list(words_df[1])
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [0]:

```
preprocessed_essays_series = pd.Series(preprocessed_essays_list)
```

After forming preprocessed_essays the process and model will be most similar to the train,test,cv model

In [0]:

```
preprocessed_essays = dataset['preprocessed_essays']
```

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(preprocessed_essays[45000:60000].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e in words_tfidf_list)
    preprocessed_essays_cv.append(sent.lower().strip())
```

```
100%|██████████| 15000/15000 [05:46<00:00, 43.32it/s]
```

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentance in tqdm(preprocessed_essays[60000:].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e in words_tfidf_list)
    preprocessed_essays_test.append(sent.lower().strip())
```

```
100%|██████████| 9999/9999 [04:30<00:00, 36.90it/s]
```

In [0]:

```
maxLen = len(max(preprocessed_essays_series[:45000], key=len).split())
maxLen
```

Out[0]:

```
234
```

In [0]:

```
from keras.preprocessing.text import Tokenizer

t = Tokenizer()
t.fit_on_texts(preprocessed_essays_series[:45000])
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(preprocessed_essays_series[:45000])
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 300
padded_docs_tfidf = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs_tfidf)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

```
Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
[[ 35 254 972 ...   0   0   0]
 [  2 562 402 ...   0   0   0]
 [195 304  30 ...   0   0   0]
 ...
 [ 19   1  83 ...   0   0   0]
 [516  24   1 ...   0   0   0]
 [ 15   3 433 ...   0   0   0]]
```

In [0]:

```
# integer encode the documents
encoded_docs = t.texts_to_sequences(preprocessed_essays_cv)
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 300
padded_docs_cv = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs_cv)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
[[    2  2445   547 ...     0     0     0]
 [    2    37    53 ...     0     0     0]
 [    2   576     1 ...     0     0     0]
 ...
 [    1     2   677 ...     0     0     0]
 [   60    88    30 ...     0     0     0]
 [ 7347  7257 10037 ...     0     0     0]]
```

In [0]:

```
# integer encode the documents
encoded_docs = t.texts_to_sequences(preprocessed_essays_test)
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 300
padded_docs_test = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs_test)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
[[   9 5224  410 ...    0    0    0]
 [  95  105    1 ...    0    0    0]
 [   9   43   97 ...    0    0    0]
 ...
 [   2   68 1438 ...    0    0    0]
 [   2  123  470 ...    0    0    0]
 [   2 3392  461 ...    0    0    0]]
```

In [0]:

```
print(padded_docs_test)
```

```
[[   9 5224  410 ...    0    0    0]
 [  95  105    1 ...    0    0    0]
 [   9   43   97 ...    0    0    0]
 ...
 [   2   68 1438 ...    0    0    0]
 [   2  123  470 ...    0    0    0]
 [   2 3392  461 ...    0    0    0]]
```

In [0]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('gdrive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))
embedding_matrix.shape
```

Out[0]:

```
(11049, 300)
```

In [0]:

```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
  if word in glove_words and word in t.word_index.keys():
    embedding_vector = model[word]
    if embedding_vector is not None:

      embedding_matrix[i] = embedding_vector
```

In [0]:

```
sequence_input = Input(shape=(max_length,), dtype='int32')
embedded_sequences = Embedding(vocab_size,output_dim=300,weights=[embedding_matrix],input_length=300,trainable=False)(sequence_input)
lstm_out = LSTM(16,return_sequences=True)(embedded_sequences)
```

In [0]:

```
flatten_layer = Flatten()
X_lstm = flatten_layer(lstm_out)
```

School_State

In [0]:

```
project_data = pd.read_csv('gdrive/My Drive/train_data.csv')
```

In [0]:

```
project_data = project_data[:69999]
```

In [0]:

```
project_data  = project_data.iloc[dataset.index,:]
```

In [0]:

```
project_data = project_data.reset_index(drop=True)
```

In [0]:

```
school_state_input = project_data['school_state'][:69999]
len(set(school_state_input))
```

Out[0]:

```
51
```

In [0]:

```
from keras.preprocessing.text import Tokenizer

t = Tokenizer()
t.fit_on_texts(school_state_input)
vocab_size = len(t.word_index) +1
encoded_docs = t.texts_to_sequences(school_state_input)

# integer encode the documents


# pad documents to a max length of 4 words
max_length = 1
padded_docs_school = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
```

In [0]:

```
input_layer_school = Input(shape=(1,))
embedding_school = Embedding(51+1,output_dim= 3, input_length=1)(input_layer_school)
flatten_school = Flatten()(embedding_school)
```

Grade

In [0]:

```
grade_category_input = project_data['project_grade_category'][:69999]
grade_category_input.nunique()
```

Out[0]:

```
4
```

In [0]:

```
z = project_data['project_grade_category'][:69999].value_counts()
z_index = list(z.index)

indices = []

for j in range(0,len(z_index)):
  index_list = []
  for i in range(0,69999):

    if(project_data['project_grade_category'][i] == z_index[j]):
      index_list.append(i)
  indices.append(index_list)


for i in range(0,len(z_index)):

  project_data['project_grade_category'][indices[i]] = i+1
```

In [0]:

```
grade_category_input = project_data['project_grade_category'][:69999]
grade_category_input = [[i] for i in list(grade_category_input)]
```

In [0]:

```
input_layer_grade = Input(shape=(1,))
embedding_grade = Embedding(input_dim=4+1,output_dim= 2, input_length=1)(input_layer_grade)
flatten_grade = Flatten()(embedding_grade)
```

In [0]:

```
padded_docs_grade = np.array(grade_category_input)
```

Clean_categories

In [0]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())
```

In [0]:

```
project_data['clean_categories'] = cat_list
```

In [0]:

```
z = project_data['clean_categories'][:69999].value_counts()
z_index = list(z.index)

indices = []

for j in range(0,len(z_index)):
  index_list = []
  for i in range(0,69999):

    if(project_data['clean_categories'][i] == z_index[j]):
      index_list.append(i)
  indices.append(index_list)


for i in range(0,len(z_index)):

  project_data['clean_categories'][indices[i]] = i+1
```

In [0]:

```
clean_categories_input = project_data['clean_categories'][:69999]
```

```
clean_categories_input = [[i] for i in list(clean_categories_input)]
```

In [0]:

```
input_layer_categories = Input(shape=(1,))
embedding_categories = Embedding(input_dim=51+1,output_dim= 3, input_length=1)
(input_layer_categories)
flatten_categories = Flatten()(embedding_categories)
```

In [0]:

```
padded_docs_categories = np.array(clean_categories_input)
```

Clean_SubCategories

In [0]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
```

In [0]:

```
project_data['clean_subcategories'].nunique()
```

Out[0]:

```
394
```

In [0]:

```
z = project_data['clean_subcategories'][:69999].value_counts()
z_index = list(z.index)

indices = []

for j in range(0,len(z_index)):
  index_list = []
  for i in range(0,69999):

    if(project_data['clean_subcategories'][i] == z_index[j]):
      index_list.append(i)
  indices.append(index_list)


for i in range(0,len(z_index)):

  project_data['clean_subcategories'][indices[i]] = i+1
```

In [0]:

```
clean_subcategories_input = project_data['clean_subcategories'][:69999]
clean_subcategories_input = [[i] for i in list(clean_subcategories_input)]
```

In [0]:

```
input_layer_subcategories = Input(shape=(1,))
embedding_subcategories = Embedding(input_dim=401+1,output_dim= 10, input_length=1)(input_layer_sub
categories)
flatten_subcategories = Flatten()(embedding_subcategories)
```

In [0]:

```
padded_docs_subcategories = np.array(clean_subcategories_input)
```

Teacher_Prefix

In [0]:

```
project_data['teacher_prefix'].nunique()
```

Out[0]:

5

In [0]:

```
#replacing nan values in pandas    https://stackoverflow.com/questions/13295735/how-can-i-replace-
all-the-nan-values-with-zeros-in-a-column-of-a-pandas-datafra
project_data['teacher_prefix'].value_counts()
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'].isnull().any()
```

Out[0]:

False

In [0]:

```
teacher_prefix_input = project_data['teacher_prefix']
```

In [0]:

```
from keras.preprocessing.text import Tokenizer

t = Tokenizer()
t.fit_on_texts(teacher_prefix_input[:69999])
vocab_size = len(t.word_index)
encoded_docs = t.texts_to_sequences(teacher_prefix_input)


# integer encode the documents


# pad documents to a max length of 4 words
max_length = 1
padded_docs_prefix = sequence.pad_sequences(encoded_docs, maxlen=max_length, padding='post')
```

In [0]:

```
input_layer_prefix = Input(shape=(1,))
embedding_prefix = Embedding(input_dim=5+1,output_dim= 2, input_length=1)(input_layer_prefix)
flatten_prefix = Flatten()(embedding_prefix)
```

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

min_max_scaler_price = preprocessing.MinMaxScaler()
price_data = min_max_scaler_price.fit_transform(dataset[['price_standardized']])
```

In [0]:

```
min_max_scaler_quantity = preprocessing.MinMaxScaler()
quantity_data = min_max_scaler_quantity.fit_transform(dataset[['quantity_standardised']])
```

In [0]:

```
min_max_scaler_digits = preprocessing.MinMaxScaler()
digits_data = min_max_scaler_digits.fit_transform(dataset[['digits_standardised']])
```

In [0]:

```
min_max_scaler_previous = preprocessing.MinMaxScaler()
previous_project_data =
min_max_scaler_previous.fit_transform(dataset[['previously_posted_projects']])
```

In [0]:

```
previous_project = Input(shape=(1,), dtype='float32')
digits = Input(shape=(1,), dtype='float32')
price = Input(shape=(1,),dtype='float32')
quantity = Input(shape=(1,),dtype='float32')

numerical_concatenate = keras.layers.concatenate([previous_project,digits,price,quantity],axis=-1)
numerical_dense = Dense(1, activation='relu')(numerical_concatenate)
```

In [0]:

```
e_concatenate = keras.layers.concatenate([X_lstm,
                                          flatten_school,
                                          flatten_grade,
                                          flatten_categories,
                                          flatten_subcategories,
                                          flatten_prefix,
                                          numerical_dense],axis=-1)


Dense_1 = Dense(16, activation='relu')(e_concatenate)
dropout_1 = Dropout(0.25)(Dense_1)
Dense_2 = Dense(32, activation='relu')(dropout_1)
dropout_2 = Dropout(0.5)(Dense_2)
Dense_3 = Dense(32, activation='relu')(dropout_2)
predictions = Dense(2, activation='sigmoid')(Dense_3)
```

In [0]:

```
input_all = []
input_all = [sequence_input,
             input_layer_school,
             input_layer_grade,
             input_layer_categories,
             input_layer_subcategories,
             input_layer_prefix,
             previous_project,
             digits,
             price,
             quantity]
```

In [0]:

```
model_2 = Model(inputs= input_all, outputs = predictions)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
opt = Adam(lr=0.01, beta_1=0.8, beta_2=0.85, decay=0.01)
model_2.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy',auroc])
```

In [0]:

```
model_2.fit([padded_docs_tfidf[:45000],
           padded_docs_school[:45000],
           padded_docs_grade[:45000],
           padded_docs_categories[:45000],
           padded_docs_subcategories[:45000],
           padded_docs_prefix[:45000],
           previous_project_data[:45000],
           digits_data[:45000],
           price_data[:45000],
           quantity_data[:45000]],y_encoded[:45000],validation_data=([padded_docs_cv,
           padded_docs_school[45000:60000],
           padded_docs_grade[45000:60000],
           padded_docs_categories[45000:60000],
           padded_docs_subcategories[45000:60000],
           padded_docs_prefix[45000:60000],
           previous_project_data[45000:60000],
           digits_data[45000:60000],
           price_data[45000:60000],
           quantity_data[45000:60000]],y_encoded[45000:60000]),epochs=3,batch_size=512)
```

```
Train on 45000 samples, validate on 15000 samples
Epoch 1/3
45000/45000 [==============================] - 121s 3ms/step - loss: 0.4324 - acc: 0.8424 - auroc:
0.6254 - val_loss: 0.3925 - val_acc: 0.8442 - val_auroc: 0.7240
Epoch 2/3
45000/45000 [==============================] - 119s 3ms/step - loss: 0.3767 - acc: 0.8510 - auroc:
0.7373 - val_loss: 0.3834 - val_acc: 0.8480 - val_auroc: 0.7457
Epoch 3/3
45000/45000 [==============================] - 119s 3ms/step - loss: 0.3485 - acc: 0.8630 - auroc:
0.7857 - val_loss: 0.3741 - val_acc: 0.8530 - val_auroc: 0.7523
```

Out[0]:

```
<keras.callbacks.History at 0x7f8b124fcd30>
```

In [0]:

```
y_predicted = model_2.predict([padded_docs_test,
               padded_docs_school[60000:],
               padded_docs_grade[60000:],
               padded_docs_categories[60000:],
               padded_docs_subcategories[60000:],
               padded_docs_prefix[60000:],
               previous_project_data[60000:],
               digits_data[60000:],
               price_data[60000:],
               quantity_data[60000:]])
```

In [0]:

```
roc_auc_score(y_encoded[60000:], y_predicted)
```

Out[0]:

```
0.7665475751277575
```

Model-3

In [0]:

```
project_data = pd.read_csv('gdrive/My Drive/train_data.csv')
```

```
project_data = project_data[:69999]
```

In [0]:

```
project_data  = project_data.iloc[index,:]
```

In [0]:

```
project_data = project_data.reset_index(drop=True)
```

In [0]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())


project_data['clean_categories'] = cat_list
```

In [0]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
```

In [0]:

```
#replacing nan values in pandas    https://stackoverflow.com/questions/13295735/how-can-i-replace-
all-the-nan-values-with-zeros-in-a-column-of-a-pandas-datafra
project_data['teacher_prefix'].value_counts()
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
```

```
project_data['teacher_prefix'].isnull().any()
```

Out[0]:

```
False
```

In [0]:

```
from sklearn.preprocessing import OneHotEncoder
onehot_encoder = OneHotEncoder()

school_encoded = onehot_encoder.fit_transform(project_data[['school_state']]).toarray()
print('outputs_encoded.shape after One Hot Encode', school_encoded.shape)


grade_encoded = onehot_encoder.fit_transform(project_data[['project_grade_category']]).toarray()
print('outputs_encoded.shape after One Hot Encode', grade_encoded.shape)


categories_encoded = onehot_encoder.fit_transform(project_data[['clean_categories']]).toarray()
print('outputs_encoded.shape after One Hot Encode', categories_encoded.shape)


subcategories_encoded =
onehot_encoder.fit_transform(project_data[['clean_subcategories']]).toarray()
print('outputs_encoded.shape after One Hot Encode', subcategories_encoded.shape)


vectorizer = CountVectorizer()
vectorizer.fit(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())

prefix_encoded = vectorizer.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",prefix_encoded.shape)
```

```
outputs_encoded.shape after One Hot Encode (69999, 51)
outputs_encoded.shape after One Hot Encode (69999, 4)
outputs_encoded.shape after One Hot Encode (69999, 49)
outputs_encoded.shape after One Hot Encode (69999, 374)
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of matrix after one hot encodig  (69999, 5)
```

In [0]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cn =
hstack((school_encoded[:69999],grade_encoded[:69999],categories_encoded[:69999],subcategories_enco
ed[:69999],prefix_encoded[:69999]))
print(X_cn.shape)
type(X_cn)
```

```
(69999, 483)
```

Out[0]:

```
scipy.sparse.coo.coo_matrix
```

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

min_max_scaler_price = preprocessing.MinMaxScaler()
price_data = min_max_scaler_price.fit_transform(dataset[['price_standardized']])

min_max_scaler_quantity = preprocessing.MinMaxScaler()
quantity_data = min_max_scaler_quantity.fit_transform(dataset[['quantity_standardised']])

min_max_scaler_digits = preprocessing.MinMaxScaler()
digits_data = min_max_scaler_digits.fit_transform(dataset[['digits_standardised']])
```

```
min_max_scaler_previous = preprocessing.MinMaxScaler()
previous_project_data =
min_max_scaler_previous.fit_transform(dataset[['previously_posted_projects']])
```

In [0]:

```
input_categorical = Input(shape=(483,), dtype='float32')
```

In [0]:

```
from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D
previous_project = Input(shape=(1,), dtype='float32')
digits = Input(shape=(1,), dtype='float32')
price = Input(shape=(1,),dtype='float32')
quantity = Input(shape=(1,),dtype='float32')

numerical_concatenate = keras.layers.concatenate([input_categorical,previous_project,digits,price,q
uantity],axis=-1)

numerical_concatenate_out = Reshape((487,1))(numerical_concatenate)

con_1d_out = Conv1D(64, 3, activation='relu')(numerical_concatenate_out)

con_1d_out = Conv1D(64, 3, activation='relu')(con_1d_out)

flatten_numerical = Flatten()(con_1d_out)
```

In [0]:

```
e_concatenate = keras.layers.concatenate([X_lstm,flatten_numerical],axis=-1)


Dense_1 = Dense(10, activation='relu')(e_concatenate)
dropout_1 = Dropout(0.4)(Dense_1)
Dense_2 = Dense(10, activation='relu')(dropout_1)
dropout_2 = Dropout(0.5)(Dense_2)
Dense_3 = Dense(20, activation='relu')(dropout_2)
predictions = Dense(2, activation='softmax')(Dense_3)
```

In [0]:

```
input_all = []
input_all = [sequence_input,
             input_categorical,
             previous_project,
             digits,
             price,
             quantity]

model_onehot = Model(inputs= input_all, outputs = predictions)
```

In [0]:

```
import tensorflow as tf
from sklearn.metrics import roc_auc_score

def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
opt = Adam(lr=0.01, beta_1=0.9, beta_2=0.999, decay=0.01)
model_onehot.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy',auroc])
```

In [0]:

```
from sklearn.preprocessing import OneHotEncoder
onehot_encoder = OneHotEncoder()
```

```
y_encoded = onehot_encoder.fit_transform(dataset[['y']]).toarray()
print('outputs_encoded.shape after One Hot Encode', y_encoded.shape)
```

```
outputs_encoded.shape after One Hot Encode (69999, 2)
```

In [0]:

```
model_onehot.fit([padded_docs_tfidf[:45000],
          X_cn.toarray()[:45000],
          previous_project_data[:45000],
          digits_data[:45000],
          price_data[:45000],
          quantity_data[:45000]],y_encoded[:45000],
          validation_data = ([padded_docs_cv,
                              X_cn.toarray()[45000:60000],
                              previous_project_data[45000:60000],
                              digits_data[45000:60000],
                              price_data[45000:60000],
                              quantity_data[45000:60000]],y_encoded[45000:60000]),
          epochs=5,batch_size=128)
```

```
Train on 45000 samples, validate on 15000 samples
Epoch 1/5
45000/45000 [==============================] - 362s 8ms/step - loss: 0.4148 - acc: 0.8479 - auroc:
0.6403 - val_loss: 0.4212 - val_acc: 0.8442 - val_auroc: 0.7168
Epoch 2/5
45000/45000 [==============================] - 372s 8ms/step - loss: 0.3911 - acc: 0.8491 - auroc:
0.7109 - val_loss: 0.4091 - val_acc: 0.8442 - val_auroc: 0.7355
Epoch 3/5
45000/45000 [==============================] - 360s 8ms/step - loss: 0.3781 - acc: 0.8500 - auroc:
0.7449 - val_loss: 0.3997 - val_acc: 0.8442 - val_auroc: 0.7464
Epoch 4/5
45000/45000 [==============================] - 360s 8ms/step - loss: 0.3639 - acc: 0.8522 - auroc:
0.7743 - val_loss: 0.3953 - val_acc: 0.8445 - val_auroc: 0.7552
Epoch 5/5
45000/45000 [==============================] - 359s 8ms/step - loss: 0.3501 - acc: 0.8528 - auroc:
0.7985 - val_loss: 0.3814 - val_acc: 0.8449 - val_auroc: 0.7592
```

Out[0]:

```
<keras.callbacks.History at 0x7f487c3dc1d0>
```

In [0]:

```
y_predicted = model_onehot.predict([padded_docs_test,
          X_cn.toarray()[60000:],
          previous_project_data[60000:],
          digits_data[60000:],
          price_data[60000:],
          quantity_data[60000:]])
```

In [0]:

```
roc_auc_score(y_encoded[60000:], y_predicted)
```

Out[0]:

```
0.7666978274361957
```

In [0]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","Epochs", "Train-Loss","Train-AUC","CV-Loss","CV-AUC","Test-AUC"]

x.add_row(["Model-1",4,0.3647,0.7516,0.3778,0.7471,0.7491])

x.add_row(["Model-3-Tfidf-Based",3,0.3485,0.7857,0.3741,0.7523,0.7665])
x.add_row(["Model-2-One-Hot-Encoding",5,0.3501,0.7985, 0.3814,0.7592,0.766699])
print(x)
```

```
+--------------------------+--------+------------+-----------+---------+--------+----------+
|          Model           | Epochs | Train-Loss | Train-AUC | CV-Loss | CV-AUC | Test-AUC |
+--------------------------+--------+------------+-----------+---------+--------+----------+
|         Model-1          |   4    |   0.3647   |   0.7516  |  0.3778 | 0.7471 |  0.7491  |
|   Model-3-Tfidf-Based    |   3    |   0.3485   |   0.7857  |  0.3741 | 0.7523 |  0.7665  |
| Model-2-One-Hot-Encoding |   5    |   0.3501   |   0.7985  |  0.3814 | 0.7592 | 0.766699 |
+--------------------------+--------+------------+-----------+---------+--------+----------+
```