Model2_LSTM

May 3, 2020

1 DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

```
Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result

How to scale current manual processes and resources to screen 500,000 projects so that they ca
<li>How to increase the consistency of project vetting across different volunteers to improve
<li>How to focus volunteer time on the applications that need the most assistance</li>
</ul>
```

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

1.1 About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`

`project_title` | Title of the project. **Examples:**
Art Will Make You Happy!
First Grade Fun
`project_grade_category` | Grade level of students for which the project is targeted. One of the following enumerated values:
Grades PreK-2
Grades 3-5
Grades 6-8
Grades 9-12
`project_subject_categories` | One or more (comma-separated) subject categories for the project from the following enumerated list of values:
Applied Learning

Care & Hunger
Health & Sports
History & Civics
Literacy & Language
Math & Science
Music & The Arts
Special Needs
Warmth
Examples:
Music & The Arts
Literacy & Language, Math & Science

`school_state` | State where school is located (Two-letter U.S. postal code). **Example:** `WY`

`project_subject_subcategories` | One or more (comma-separated) subject subcategories for the project. **Examples:**

Literacy
Literature & Writing, Social Sciences

`project_resource_summary` | An explanation of the resources needed for the project. **Example:**

My students need hands on literacy materials to manage sensory needs!

`project_essay_1` | First application essay
`project_essay_2` | *Second application essay* `project_essay_3` | Third application essay
`project_essay_4` | *Fourth application essay* `project_submitted_datetime` | Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id` | A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix` | Teacher's title. One of the following enumerated values:

nan
Dr.
Mr.
Mrs.
Ms.
Teacher.

`teacher_number_of_previously_posted_projects` | Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section Notes on the Essay Data for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`

Feature	Description
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

1.1.1 Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

project_essay_1: "Introduce us to your classroom"

project_essay_2: "Tell us more about your students"

project_essay_3: "Describe how your students will use the materials you're requesting"

project_essay_3: "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

project_essay_1: "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

project_essay_2: "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

```
In [0]: !pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69dd2a9cbff0
     || 71kB 3.4MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from chart_s
```

```
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from chart_studio
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from chart_stu
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from req
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (fro
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

```
In [0]: import chart_studio.plotly as py
        import plotly.graph_objs as go
```

```
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
```

```
In [0]: %tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

1.2 Import File from local drive to Colab Notebook

```
In [0]: # Load the Drive helper and mount
        from google.colab import drive

        # This will prompt for authorization.
        drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6

Enter your authorization code:
··········
Mounted at /content/drive
```

1.3 1.1 Reading Data

```
In [0]: import io
        resource_data=pd.read_csv('/content/drive/My Drive/LSTM Assignment/resources.csv')
        project_data=pd.read_csv('/content/drive/My Drive/LSTM Assignment/train_data.csv')
        #project_data =project_data.sample(n=1000)
```

```
In [0]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_i
        project_data = pd.merge(project_data, price_data, on='id', how='left')
```

1.4 1.2 preprocessing of `project_subject_categories`

```
In [0]: catogories = list(project_data['project_subject_categories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

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

```
            if 'The' in j.split(): # this will split each of the catogory based on space "
                j=j.replace('The','') # if we have the words "The" we are going to replace
            j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
            temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing sp
            temp = temp.replace('&','_') # we are replacing the & value into
        cat_list.append(temp.strip())

    project_data['clean_categories'] = cat_list
    project_data.drop(['project_subject_categories'], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in project_data['clean_categories'].values:
        my_counter.update(word.split())

    cat_dict = dict(my_counter)
    sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.5 1.3 preprocessing of project_subject_subcategories

```
In [0]: sub_catogories = list(project_data['project_subject_subcategories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

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

        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", "Warmt
                if 'The' in j.split(): # this will split each of the catogory based on space "
                    j=j.replace('The','') # if we have the words "The" we are going to replace
                j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
                temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing sp
                temp = temp.replace('&','_')
            sub_cat_list.append(temp.strip())

        project_data['clean_subcategories'] = sub_cat_list
        project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

        # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        my_counter = Counter()
        for word in project_data['clean_subcategories'].values:
            my_counter.update(word.split())

        sub_cat_dict = dict(my_counter)
```

```
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[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]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
            'won', "won't", 'wouldn', "wouldn't"]
```

In [0]:
```
# Combining all the above stundents
def Text_cleaner(data):

    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentance in tqdm(data.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.lower() not in stopwords)
            preprocessed_essays.append(sent.lower().strip())
        return preprocessed_essays
```

```
In [0]: # merge two column text dataframe:
        project_data["essay"] = project_data["project_essay_1"].map(str) +\
                                project_data["project_essay_2"].map(str) + \
                                project_data["project_essay_3"].map(str) + \
                                project_data["project_essay_4"].map(str)
```

```
In [0]: project_data['essay']=Text_cleaner(project_data['essay'])
```

```
100%|| 109248/109248 [00:51<00:00, 2131.95it/s]
```

```
In [0]: #https://stackoverflow.com/questions/29763620/how-to-select-all-columns-except-one-col
        X=project_data.loc[:, project_data.columns != 'project_is_approved']
        y=project_data['project_is_approved']
        X.shape
```

```
Out[0]: (109248, 19)
```

```
In [0]: X.head()
```

```
Out[0]:                                               essay  ... quantity
        0  students english learners working english seco...  ...       23
        1  students arrive school eager learn polite gene...  ...        1
        2  true champions not always ones win guts mia ha...  ...       22
        3  work unique school filled esl english second l...  ...        4
        4  second grade classroom next year made around 2...  ...        4

        [5 rows x 9 columns]
```

1.6 Splitting data into Test,Train,CV

```
In [0]: from sklearn.model_selection import train_test_split

        X_train,X_test,y_train, y_test=train_test_split(X, y, test_size=0.2)
        X_train,X_cv,y_train,y_cv=train_test_split(X_train, y_train, test_size=0.2)
        print(X_train.shape)
        print(X_test.shape)

        print(y_train.shape)
        print(y_test.shape)
```

```
(69918, 9)
(21850, 9)
(69918,)
(21850,)
```

1.7 TFIDF Vectorize the Data

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        vectorizer = TfidfVectorizer(min_df=6,use_idf=True)
        vectorizer.fit(X_train['essay'])

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_essay=vectorizer.transform(X_train['essay'].values)
        X_test_essay=vectorizer.transform(X_test['essay'].values)
        X_cv_essay=vectorizer.transform(X_cv['essay'].values)


        print(X_train_essay.shape)
        print(X_test_essay.shape)
        print(X_cv_essay.shape)
```

```
(69918, 17214)
(21850, 17214)
(17480, 17214)
```

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        #Fit the TFIDF vectorizer to the train data
        vect = TfidfVectorizer()
        vect.fit_transform(X_train['essay'])
        #Get the features names and their corresponding IDF scores
        words = vect.get_feature_names()
        idf_words = vect.idf_
        #Map the words and their idf_ scores in a disctionary
        dict_word_idf_ = dict(zip(words, idf_words))
        print("Total number of unique words present originally:", len(words))
```

```
Total number of unique words present originally: 47205
```

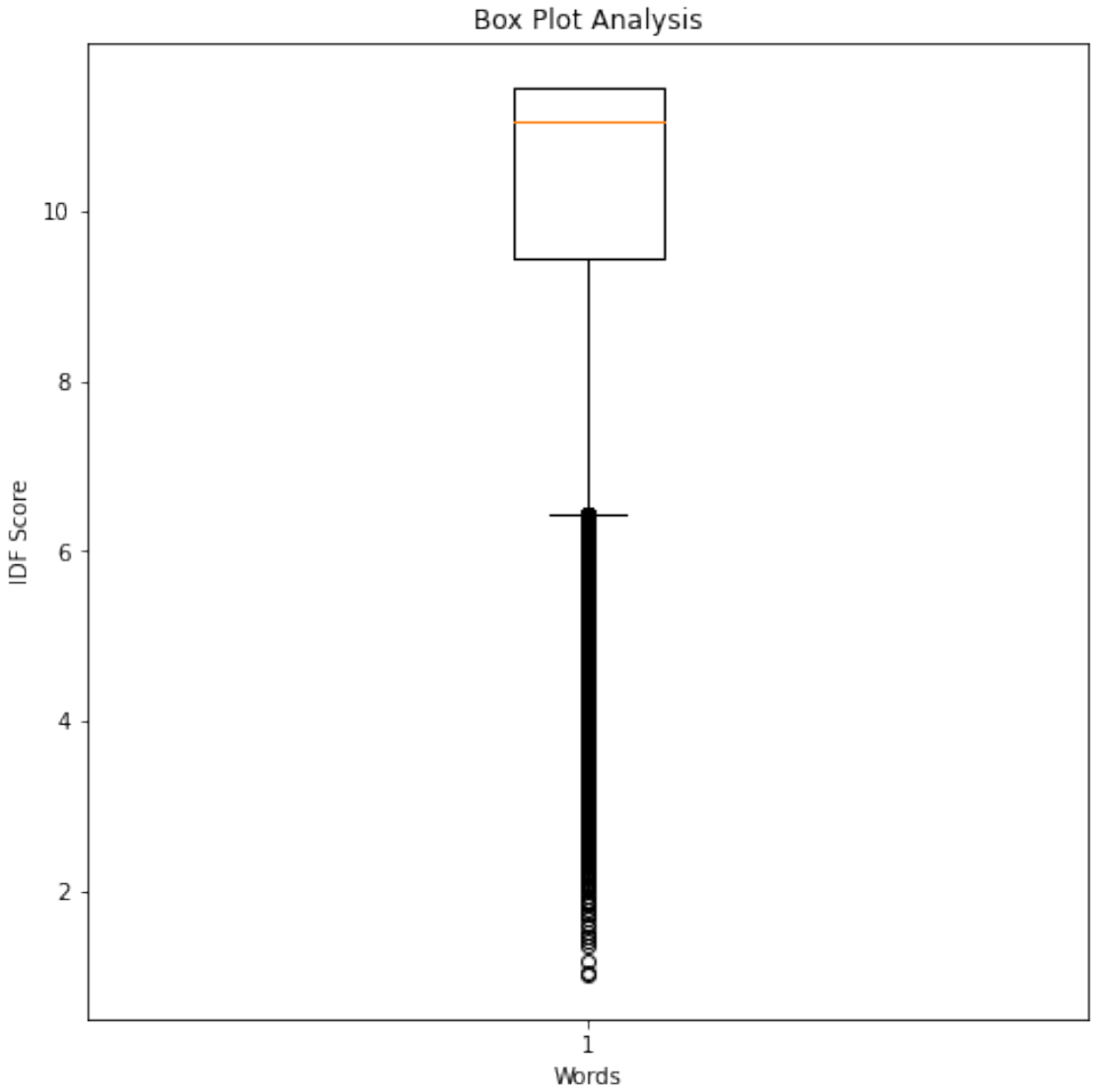
```
In [0]: maximum = max(dict_word_idf_, key=dict_word_idf_.get) # Just use 'min' instead of 'max
        print(maximum, dict_word_idf_[maximum])
```

```
001 11.4619455276077
```

```
In [0]: #https://stackoverflow.com/questions/26871866/print-highest-value-in-dict-with-key
        minimum = min(dict_word_idf_, key=dict_word_idf_.get) # Just use 'min' instead of 'max
        print(minimum, dict_word_idf_[minimum])
```

```
students 1.007955011652819
```

```
In [0]: import matplotlib.pyplot as plt
        plt.figure(figsize=(8,8))
        plt.boxplot(idf_words)
        plt.title('Box Plot Analysis')
        plt.xlabel('Words')
        plt.ylabel('IDF Score')
        plt.show()
```




```
In [0]: #https://stackoverflow.com/questions/34232190/scikit-learn-tfidfvectorizer-how-to-get-
        removed_wordlist = []
```

```
        for word in list(dict_word_idf_.keys()):
          if(dict_word_idf_[word] < 2 or dict_word_idf_[word] > 11):
            removed_wordlist.append(word)
          else:
            continue
        print("Number of words to be removed: ",len(removed_wordlist))
```

```
Number of words to be removed:  23839
```

```
In [0]: def remove_from_text(list_of_sentences):
            """This function will be used to remove words from text data"""
            processed_text = []
            for sentence in tqdm(list_of_sentences):
                sent = ' '.join(word for word in sentence.split() if word not in removed_wordli
                processed_text.append(sent)
            return processed_text

        X_train['essay'] = remove_from_text(X_train.essay.values)
        X_test['essay'] = remove_from_text(X_test.essay.values)
        X_cv['essay'] = remove_from_text(X_cv.essay.values)
```

1.8 Load the GloVe from text

```
In [0]: from numpy import asarray
        def get_glove():
          embeddings_index = dict()
          f = open('/content/glove.6B.300d.txt')
          for line in tqdm(f):
            values = line.split()
            word = values[0]
            coefs = asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
          f.close()
          return embeddings_index
```

```
In [0]: embeddings_index=get_glove()
```

```
400000it [00:27, 14396.75it/s]
```

```
In [0]: from keras.preprocessing.text import Tokenizer
        from keras.preprocessing.sequence import pad_sequences
        from numpy import zeros



        #def word_embedding(docs,embeddings_index):
        glove_words =  set(embeddings_index.keys())
```

```
sentence_vectors = []
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(X_train['essay'].values)
vocab_size = len(t.word_index) + 1
# integer encode the documents
X_train_encoded_docs = t.texts_to_sequences(X_train['essay'].values)
X_test_encoded_docs = t.texts_to_sequences(X_test['essay'].values)
X_cv_encoded_docs = t.texts_to_sequences(X_cv['essay'].values)

# pad documents to a max length of 300 words
max_length = 300
X_train_padded_docs = pad_sequences(X_train_encoded_docs, maxlen=max_length, padding='
X_test_padded_docs = pad_sequences(X_test_encoded_docs, maxlen=max_length, padding='pos
X_cv_padded_docs = pad_sequences(X_cv_encoded_docs, maxlen=max_length, padding='post')

embedding_matrix = zeros((vocab_size, max_length))
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

print(X_train_padded_docs.shape)
print(X_test_padded_docs.shape)
print(X_cv_padded_docs.shape)
```

```
Using TensorFlow backend.
```

```
(69918, 300)
(21850, 300)
(17480, 300)
```

```
In [0]: essay_input = Input(shape=(len(X_train_padded_docs[0]),), name='essay_input')
        essay_input_1 = Embedding(input_dim=vocab_size,output_dim=300, input_length=len(X_trai
        essay_input_1 = LSTM(32,return_sequences=True)(essay_input_1)
        essay_input_1 = Flatten()(essay_input_1)
```

1.8.1 Vectorize the Features

1.9 Categorical data: clean_categories

1.9.1 Building train, test and validation data

```
In [0]: # we use count vectorizer to convert the values into one
        from sklearn.feature_extraction.text import CountVectorizer
```

```
        vectorizer = CountVectorizer(lowercase=False, binary=True)
        vectorizer.fit(X['clean_categories'].values)

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_cat=vectorizer.transform(X_train['clean_categories'].values)
        X_test_cat=vectorizer.transform(X_test['clean_categories'].values)
        X_cv_cat=vectorizer.transform(X_cv['clean_categories'].values)

        print(vectorizer.get_feature_names())
        print(X_train_cat.shape)
        print(X_test_cat.shape)
        print(X_cv_cat.shape)
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy_Language', 'Mat
(69918, 9)
(21850, 9)
(17480, 9)
```

```
In [0]: categories_input = Input(shape=(X_train_cat.shape[1],), name='categories_input')
        categories_input_1= Embedding(input_dim=X_train_cat.shape[1],output_dim=64, input_lengt
        categories_input_1 = Flatten()(categories_input_1)
```

1.10 Categorical data: clean_subcategories

1.10.1 Building train, test and validation data

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer(lowercase=False, binary=True)
        vectorizer.fit(X['clean_subcategories'].values)

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_sub_cat=vectorizer.transform(X_train['clean_subcategories'].values)
        X_test_sub_cat=vectorizer.transform(X_test['clean_subcategories'].values)
        X_cv_sub_cat=vectorizer.transform(X_cv['clean_subcategories'].values)

        print(vectorizer.get_feature_names())
        print(X_train_sub_cat.shape)
        print(X_test_sub_cat.shape)
        print(X_cv_sub_cat.shape)
```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government', 'College_CareerPr
(69918, 30)
(21850, 30)
(17480, 30)
```

```
In [0]: sub_categories_input = Input(shape=(X_train_sub_cat.shape[1],), name='sub_categories_in
        sub_categories_input_1= Embedding(input_dim=X_train_sub_cat.shape[1],output_dim=64, inp
        sub_categories_input_1 = Flatten()(sub_categories_input_1)
```

1.11 Categorical data: project_grade_category

1.11.1 Building train, test and validation data

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer(lowercase=False, binary=True)
        vectorizer.fit(X['project_grade_category'].values)

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_pro_grd=vectorizer.transform(X_train['project_grade_category'].values)
        X_test_pro_grd=vectorizer.transform(X_test['project_grade_category'].values)
        X_cv_pro_grd=vectorizer.transform(X_cv['project_grade_category'].values)

        print(vectorizer.get_feature_names())
        print(X_train_pro_grd.shape)
        print(X_test_pro_grd.shape)
        print(X_cv_pro_grd.shape)
```

```
['Grades35', 'Grades68', 'Grades912', 'GradesPreK2']
(69918, 4)
(21850, 4)
(17480, 4)
```

```
In [0]: proj_grade_input = Input(shape=(X_train_pro_grd.shape[1],), name='proj_grade_input')
        proj_grade_input_1= Embedding(input_dim=X_train_pro_grd.shape[1],output_dim=64, input_
        proj_grade_input_1 = Flatten()(proj_grade_input_1)
```

1.12 Categorical data: school_state

1.12.1 Building train, test and validation data

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer(lowercase=False, binary=True)
        vectorizer.fit(X['school_state'].values)

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_skl_st=vectorizer.transform(X_train['school_state'].values)
        X_test_skl_st=vectorizer.transform(X_test['school_state'].values)
        X_cv_skl_st=vectorizer.transform(X_cv['school_state'].values)

        print(vectorizer.get_feature_names())
        print(X_train_skl_st.shape)
        print(X_test_skl_st.shape)
        print(X_cv_skl_st.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN
(69918, 51)
(21850, 51)
(17480, 51)
```

```
In [0]: school_state_input = Input(shape=(X_train_skl_st.shape[1],), name='school_state_input')
        school_state_input_1= Embedding(input_dim=X_train_skl_st.shape[1],output_dim=64, input_
        school_state_input_1 = Flatten()(school_state_input_1)
```

1.13 Categorical data: teacher_prefix

1.13.1 Building train, test and validation data

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer(lowercase=False, binary=True)
        vectorizer.fit(X['teacher_prefix'].values)

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_tch=vectorizer.transform(X_train['teacher_prefix'].values)
        X_test_tch=vectorizer.transform(X_test['teacher_prefix'].values)
        X_cv_tch=vectorizer.transform(X_cv['teacher_prefix'].values)

        print(vectorizer.get_feature_names())
        print(X_train_tch.shape)
        print(X_test_tch.shape)
        print(X_cv_tch.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
(69918, 5)
(21850, 5)
(17480, 5)
```

```
In [0]: tch_input = Input(shape=(X_train_tch.shape[1],), name='tch_input')
        tch_input_1= Embedding(input_dim=X_train_tch.shape[1],output_dim=64, input_length=X_tra
        tch_input_1 = Flatten()(tch_input_1)
```

1.14 price

1.14.1 Building train, test and validation data

```
In [0]: from sklearn.preprocessing import MinMaxScaler

        scaler = MinMaxScaler()
        scaler.fit(X_train['price'].values.reshape(-1,1))
        X_train_price=scaler.transform(X_train['price'].values.reshape(-1,1))
        X_test_price=scaler.transform(X_test['price'].values.reshape(-1,1))
        X_cv_price=scaler.transform(X_cv['price'].values.reshape(-1,1))


        print(X_train_price.shape)
        print(X_test_price.shape)
        print(X_cv_price.shape)
```

```
(69918, 1)
(21850, 1)
(17480, 1)
```

1.15 quantity

1.15.1 Building train and validation data

```
In [0]: scaler = MinMaxScaler()
        scaler.fit(X_train['quantity'].values.reshape(-1,1))
        X_train_quantity=scaler.transform(X_train['quantity'].values.reshape(-1,1))
        X_test_quantity=scaler.transform(X_test['quantity'].values.reshape(-1,1))
        X_cv_quantity=scaler.transform(X_cv['quantity'].values.reshape(-1,1))


        print(X_train_quantity.shape)
        print(X_test_quantity.shape)
        print(X_cv_quantity.shape)
```

```
(69918, 1)
(21850, 1)
(17480, 1)
```

1.16 teacher_number_of_previously_posted_projects

1.16.1 Building train, test and validation data

```
In [0]: scaler = MinMaxScaler()
        scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
        X_train_pre_proj=scaler.transform(X_train['teacher_number_of_previously_posted_projects
        X_test_pre_proj=scaler.transform(X_test['teacher_number_of_previously_posted_projects']
        X_cv_pre_proj=scaler.transform(X_cv['teacher_number_of_previously_posted_projects'].val


        print(X_train_pre_proj.shape)
        print(X_test_pre_proj.shape)
        print(X_cv_pre_proj.shape)
```

```
(69918, 1)
(21850, 1)
(17480, 1)
```

```
In [0]: X_train_numerals=np.concatenate((X_train_pre_proj,X_train_quantity,X_train_price),axis=
        X_test_numerals=np.concatenate((X_test_pre_proj,X_test_quantity,X_test_price),axis=1)
        X_cv_numerals=np.concatenate((X_cv_pre_proj,X_cv_quantity,X_cv_price),axis=1)
```

```
print(X_train_numerals.shape)
print(X_test_numerals.shape)
print(X_cv_numerals.shape)
```

```
(69918, 3)
(21850, 3)
(17480, 3)
```

```
In [0]: numeral_input=Input(shape=(X_train_numerals.shape[1],),name='numeral_input')
        numeral_input_1 = Dense(input_dim=3,output_dim=128)(numeral_input)
```

1.17 Concatenate

```
In [0]: #merge them two
        x = concatenate([essay_input_1,categories_input_1,sub_categories_input_1,proj_grade_inp
```

1.17.1 Defining a custom metric AUC

```
In [0]: import tensorflow as tf
        from sklearn.metrics import roc_auc_score

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

          except ValueError:
            pass
```

1.18 Applying the Best found model

```
In [0]: # !pip install -q tf-nightly-2.0-preview
        # if you want to use the tf2.0 please uncomment the above line
        # Load the TensorBoard notebook extension
        # there are other ways of doing this: https://www.dlology.com/blog/quick-guide-to-run-
        %load_ext tensorboard
```

```
In [0]: # Clear any logs from previous runs
        !rm -rf ./logs/
```

```
In [0]: import tensorflow as tf
        from time import time
        import datetime
        from tensorflow.python.keras.callbacks import TensorBoard, EarlyStopping
        from keras.callbacks import TensorBoard
        from keras.callbacks import ModelCheckpoint
        #tensorboard = TensorBoard(log_dir="logs/{}".format(time))
        log_dir="logs\\fit\\" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
        tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=(
```

```
        filepath="model3.h5"
        checkpoint = ModelCheckpoint(filepath, monitor='val_auroc', verbose=1, save_best_only='
```

In [0]:
```
        from keras.layers import Input, Embedding, LSTM, Dense,Flatten,concatenate,Dropout,Bat
        from keras.models import Model
        from keras.callbacks import EarlyStopping


        # And finally we add the main logistic regression layer
        output =Dense(64, activation='relu')(x)
        output =BatchNormalization()(output)
        output =Dropout(0.5)(output)
        output =Dense(32, activation='relu')(output)
        output =BatchNormalization()(output)
        output =Dropout(0.5)(output)
        output =Dense(16, activation='relu')(output)
        output =BatchNormalization()(output)
        output =Dropout(0.5)(output)
        output =Dense(1, activation='sigmoid',name='output')(output)

        #defines model with multiple input one output
        model = Model(inputs=[essay_input, categories_input,sub_categories_input,proj_grade_inp

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

        # simple early stopping
        es = EarlyStopping(monitor='val_loss', patience=4,verbose=0, mode='min')

        model.fit([X_train_padded_docs, X_train_cat,X_train_sub_cat,X_train_pro_grd,X_train_sk
                  epochs=30,
                  batch_size=1000,
                  validation_data=([X_cv_padded_docs, X_cv_cat,X_cv_sub_cat,X_cv_pro_grd,X_cv_
                  verbose=1,
                  callbacks=[es,tensorboard_callback,checkpoint])
```

```
Train on 69918 samples, validate on 17480 samples
Epoch 1/30
69918/69918 [==============================] - 23s 328us/step - loss: 0.7994 - accuracy: 0.573

Epoch 00001: val_auroc improved from -inf to 0.49061, saving model to model3.h5
Epoch 2/30
69918/69918 [==============================] - 22s 310us/step - loss: 0.6333 - accuracy: 0.718

Epoch 00002: val_auroc improved from 0.49061 to 0.53992, saving model to model3.h5
Epoch 3/30
69918/69918 [==============================] - 21s 307us/step - loss: 0.5431 - accuracy: 0.812

Epoch 00003: val_auroc did not improve from 0.53992
```

```
Epoch 4/30
69918/69918 [==============================] - 21s 304us/step - loss: 0.4809 - accuracy: 0.836

Epoch 00004: val_auroc improved from 0.53992 to 0.68386, saving model to model3.h5
Epoch 5/30
69918/69918 [==============================] - 21s 305us/step - loss: 0.4419 - accuracy: 0.842

Epoch 00005: val_auroc improved from 0.68386 to 0.70174, saving model to model3.h5
Epoch 6/30
69918/69918 [==============================] - 21s 303us/step - loss: 0.4276 - accuracy: 0.844

Epoch 00006: val_auroc improved from 0.70174 to 0.72242, saving model to model3.h5
Epoch 7/30
69918/69918 [==============================] - 21s 307us/step - loss: 0.4210 - accuracy: 0.845

Epoch 00007: val_auroc improved from 0.72242 to 0.72581, saving model to model3.h5
Epoch 8/30
69918/69918 [==============================] - 21s 307us/step - loss: 0.4119 - accuracy: 0.846

Epoch 00008: val_auroc improved from 0.72581 to 0.72946, saving model to model3.h5
Epoch 9/30
69918/69918 [==============================] - 21s 304us/step - loss: 0.4043 - accuracy: 0.847

Epoch 00009: val_auroc did not improve from 0.72946
Epoch 10/30
69918/69918 [==============================] - 21s 305us/step - loss: 0.3993 - accuracy: 0.848

Epoch 00010: val_auroc improved from 0.72946 to 0.73156, saving model to model3.h5
Epoch 11/30
69918/69918 [==============================] - 21s 306us/step - loss: 0.3985 - accuracy: 0.847

Epoch 00011: val_auroc did not improve from 0.73156
Epoch 12/30
69918/69918 [==============================] - 21s 303us/step - loss: 0.3899 - accuracy: 0.848

Epoch 00012: val_auroc did not improve from 0.73156
Epoch 13/30
69918/69918 [==============================] - 21s 306us/step - loss: 0.3845 - accuracy: 0.849

Epoch 00013: val_auroc did not improve from 0.73156


Out[0]: <keras.callbacks.callbacks.History at 0x7fbf91ab5fd0>

In [0]: #%tensorboard --logdir .
        %tensorboard --logdir logs\\fit\\20200502-200242

<IPython.core.display.HTML object>
```

```
In [1]: from IPython.display import Image
        from IPython.core.display import HTML
        Image(url= "https://imgur.com/a/71UWi22.jpg")
```

```
Out[1]: <IPython.core.display.Image object>
```