In “analisys” folder :

- Analyzer.py : script used for analyzing mood of user
- Csvtojson.py : script that permit to transform file csv in this case analisys/tmdb_movies_data.csv to tmdb_movies.json file that contain information about film without sentmenti score
- sentAnalyzer.py: script that create moviesent.json file that contain information about films and sentiment score from 0 (negati sentiment)to 1 (positive sentiment) obtained after analysing the sentiment of description of film , and genres.json that containg every differet genres present in our movie list
- word2vect.py : script that contain implemented word2vect module for analysing “sentiment” of movies
- aclImdb folder : folder used for obtain movie_data.csv
- makecsv.py : script used for creating movie_data.csv

In “scripts” folder :

- downloadmodule.py: script used for dowunload some fundemental module necessary for correct working of word2vect.py

Dataset:

- tmdb_movies_data.csv: This dataset downloaded from:https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv contain the list of movies that we are showing to user and that we are analiysing and giving a score of sentiment from 0 to 1 , and after analysis we are creating moviesent.json that is the list of movie with sentiment score of film thi file contain this following attribute for each movie : title, year,director,cast,link,ratings,genres,duration,overwiew,sentiment

- analisys/aclImdb(movie_data.csv): The IMDB movie review set can be downloaded from http://ai.stanford.edu/~amaas/data/sentiment/ . This dataset for binary sentiment classification contains set of 25,000 highly polar movie reviews for training, and 25,000 for testing. The dataset after initial pre-processing (with makecsv.py) is saved to movie_data.csv file. First we load the IMDb dataset, the text reviews are labelled as 1 or 0 for positive and negative sentiment respectively , but for timing reason we are using directly movie_data.csv that is 65.9 mb so it is fast if we use direcly movie_data.csv file to save time and don’t wait the preposcessing script (makecsv.py)

Sentimenent analysy :
for implemetattion of sentiment analysis we are using two different method

- Frst metod is using vaderSentiment python tool this is the description : VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains.VADER uses a combination of A sentiment lexicon is a list of lexical features (e.g., words) which are generally labelled according to their semantic orientation as either positive or negative.VADER has been found to be quite successful when dealing with social media texts, NY Times editorials, movie reviews, and product reviews. This is because VADER not only tells **about** the Positivity and Negativity score but also tells us about **how positive or negative a sentiment is**.
  And very importa thing that It is fully open-sourced under the MIT License.
  For example of usage and for more information we seggest following links :
    - .https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f in this aricle we can find lot of usefull information and example of usage
    - .https://pypi.org/project/vaderSentiment/ this is the official link of python liblrary
    - Github : https://github.com/cjhutto/vaderSentiment

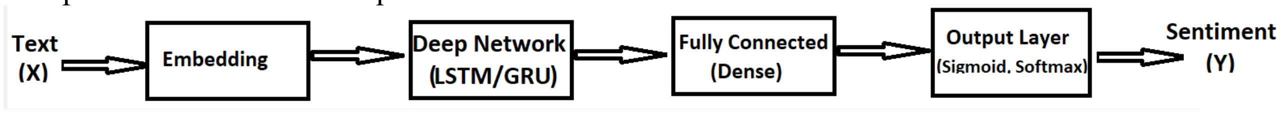
This is our script whit this tool

```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords

#nltk.download('stopwords')
#set(stopwords.words('english'))


def analyzer(text):
    stop_words = stopwords.words('english')
    text1 = text
    processed_doc1 = ' '.join([word for word in text1.split() if word not in
stop_words])
    sa = SentimentIntensityAnalyzer()
    dd = sa.polarity_scores(text=processed_doc1)
    compound = round((1 + dd['compound']) / 2, 2)
    return compound
```

Really essensial script when we use analyzer with passing a text the function return a score of sentiment of this text
we are usign this method to get the sentiment of user because this metod is much more fast than second one that is used to get a score of our movies list

- The second method is using Deep Learning model for sentiment classification using word2vec model as a per-trained embedding for sentiment classification

theDeep learning text classification model architectures generally consist of the following components connected in sequence:



Now few definition of this levels:

- Embedding Layer:
  Word Embedding is a representation of text where words that have the same meaning have a similar representation. In other words it represents words in a coordinate system where related words, based on a corpus of relationships, are placed closer together. In the deep learning frameworks such as TensorFlow, Keras, this part is usually handled by an embedding layer which stores a lookup table to map the words represented by numeric indexes to their dense vector representations.
- Deep Network:
  Deep network takes the sequence of embedding vectors as input and converts them to a compressed representation. The compressed representation effectively captures all the information in the sequence of words in the text. The deep neywrok part is usually an RNN or some forms of it like LSTM/GRU. The dropout is added to overcome the tendency to overfit, a very common problem with RNN based networks. Please refer here for detailed discussion on LSTM,GRU.
- Fully Connected Layer
  The **fully connected layer** takes the deep representation from the RNN/LSTM/GRU and transforms it into the final output classes or class scores. This component is comprised of fully connected layers along with batch normalization and optionally dropout layers for regularization.
- Output Layer
  Based on the problem at hand, this layer can have either **Sigmoid** for binary classification or **Softmax** for both binary and multi classification output.

Lets see part of code used for implement this method

- **For implemtet word2vec Embedding:**
  We will use the Gensim implementation of Word2Vec. The first step is to prepare the text corpus for learning the embedding by creating word tokens, removing punctuation, removing stop words etc. The word2vec algorithm processes documents sentence by sentence:

```
review_lines = list()
lines = df['review'].values.tolist()

for line in lines:
    tokens = word_tokenize(line)
    # convert to lower case
    tokens = [w.lower() for w in tokens]
    # remove punctuation from each word
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    words = [word for word in stripped if word.isalpha()]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    review_lines.append(words)
```

we have 50000 review lines in our text corpus. Gensim's Word2Vec API requires some parameters for initialization.like :
-sentences :List of sentences; here we pass the list of review sentences.
-size :The number of dimensions in which we wish to represent our word. This is the size of the word vector.
-min_count :Word with frequency greater than min_count only are going to be included into the model. Usually, the bigger and more extensive your text, the higher this number can be.
-window :Only terms that occur within a *window*-neighborhood of a term, in a sentence, are associated with it during training. The usual value is 4 or 5.
-workers–:Number of threads used in training parallelization, to speed up training

```
# train word2vec model
model = gensim.models.Word2Vec(sentences=review_lines, size=EMBEDDING_DIM,
window=5, workers=4, min_count=1)
# vocab size
words = list(model.wv.vocab)
```

After we train the model on our IMDb dataset, it builds a vocabulary size = 134156

-The next step is to use the word embeddings directly in the embedding layer in our sentiment classification model. we can save the model to be used later.

```
# save model in ASCII (word2vec) format
filename = 'imdb_embedding_word2vec.txt'
model.wv.save_word2vec_format(filename, binary=False)
```

Since we have already trained word2vec model with IMDb dataset, we have the word embeddings ready to use. The next step is to load the word embedding as a directory of words to vectors. The word embedding was saved in file imdb_embedding_word2vec.txt. Let us extract the word embeddings from the stored file.

```

embeddings_index = {}
f = open(os.path.join('', 'imdb_embedding_word2vec.txt'), encoding="utf-8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:])
    embeddings_index[word] = coefs
f.close()
```

- The next step is to convert the word embedding into tokenized vector. Recall that the review documents are integer encoded prior to passing them to the Embedding layer. The integer maps to the index of a specific vector in the embedding layer. Therefore, it is important that we lay the vectors out in the Embedding layer such that the encoded words map to the correct vector

```

# vectorize the text samples into a 2D integer tensor
tokenizer_obj = Tokenizer()
tokenizer_obj.fit_on_texts(review_lines)
sequences = tokenizer_obj.texts_to_sequences(review_lines)

# pad sequences
word_index = tokenizer_obj.word_index

review_pad = pad_sequences(sequences, maxlen=max_length)
sentiment = df['sentiment'].values
```

For thi step we are using:

```
tensorflow.python.keras.preprocessing.text import Tokenizer
```

Now we will map embeddings from the loaded word2vec model for each word to the tokenizer_obj.word_index vocabulary and create a matrix with of word vectors

```

num_words = len(word_index) + 1
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))

for word, i in word_index.items():
    if i > num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

We are now ready with the trained embedding vector to be used directly in the embedding layer. In the below code we are using the embedding_matrix as input to the Embedding layer and setting trainable = False, since the embedding is already learned

```
model = Sequential()
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=max_length,
                            trainable=False)
model.add(embedding_layer)
model.add(GRU(units=32, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

-To train the sentiment classification model, we use VALIDATION_SPLIT= 0.2, you can vary this to see effect on the accuracy of the model**.**

```

VALIDATION_SPLIT = 0.2
# split the data into a training set and a validation set
indices = np.arange(review_pad.shape[0])
np.random.shuffle(indices)
review_pad = review_pad[indices]
sentiment = sentiment[indices]
num_validation_samples = int(VALIDATION_SPLIT * review_pad.shape[0])

X_train_pad = review_pad[:-num_validation_samples]
y_train = sentiment[:-num_validation_samples]
X_test_pad = review_pad[-num_validation_samples:]
y_test = sentiment[-num_validation_samples:]
```

Now let us train the model on training set and cross validate on test set. We can see from below training epochs that the model after each epoch is improving the accuracy. After a few epochs we reach validation accuracy of around 75%

```
model.fit(X_train_pad, y_train, batch_size=128, epochs=25,
validation_data=(X_test_pad, y_test), verbose=2)
```

The next step is saving our model in "model.h5" file and tokenized vector into "tokenizer.pickle"

```
#save model and token
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer_obj, handle, protocol=pickle.HIGHEST_PROTOCOL)

model.save("model.h5")
```

The next step is use our model to give the score of sentiment for movies :
First we are loading token vector and our model and after with function predict of our model
We are giving for each movies a score of sentimet from 0 to 1

```python
with open('tokenizer.pickle', 'rb') as handle:
        tokenizer_obj = pickle.load(handle)

model = load_model('model.h5')

# csvtojson()
genres = []
with open('analisys/tmdb_movie.json') as infile:
    movie = json.load(infile)
text = []
movie2 = []
g = 0
with open('moviesent.json', 'w') as f:
    for i in range(0, len(movie)):
        if movie[i]['overview'] is not None:
            sentiment = []
            text = movie[i]['overview']
            text_tokens = tokenizer_obj.texts_to_sequences(text)
            tex_tokens_pad = pad_sequences(text_tokens, maxlen=35)
            sentiment = model.predict(x=tex_tokens_pad)
            movie2.append({"title": movie[i]['original_title'], "year":
movie[i]['release_year'],
                           "director": movie[i]['director'], "cast":
movie[i]['cast'], "link": movie[i]['homepage'],
                           "ratings": movie[i]['vote_average'], "genres":
movie[i]['genres'],
                           "duration": movie[i]['runtime'], "overview":
movie[i]['overview'],
                           "sentiment": float(str(sentiment[0][0]))})

    # print(sentiment)
    json.dump(movie2, f, indent=2)
```

after getting a score of sentiment we are creating a moviesent.json that contains the list of movie and sentiment score as you can see :

```json
{
  "title": "Jurassic World",
  "year": 2015,
  "director": "Colin Trevorrow",
  "cast": "Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vincent D'Onofrio|Nick
Robinson",
  "link": "http://www.jurassicworld.com/",
  "ratings": 6.5,
  "genres": "Action|Adventure|Science Fiction|Thriller",
  "duration": 124,
  "overview": "Twenty-two years after the events of Jurassic Park, Isla Nublar
now features a fully functioning dinosaur theme park, Jurassic World, as
originally envisioned by John Hammond.",
  "sentiment": 0.4816491
},
```

Usefull links for this module can be :

- For gensim wor2vect: : https://radimrehurek.com/gensim/models/word2vec.html
- For Keras: https://keras.io/api/
- guide used for understanding how to implement word2vect : https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456