Anand Arjan Jhamnani anandarj 50320027
Abhishek Dalvi adalvi 50320110
Harishkandan Somasundarram harsihk2 50313808

Changes made in the VM:
Java version was changed using the following command:
sudo update-alternatives --config java

import findspark
and
findspark.init()
Was added to the codes.
SPARK_HOME environment variable was added to the bashrc file

Part 1:

Base model

Steps for preprocessing:

1) Tokenizing
   The first important step is to tokenize the sentences into a list of word. We've used a Regex Tokenizer here.
2) Stopwords
   Words such as 'the', 'a', are useless for our classification even if they appear a lot in the document and are hence removed
3) Vectorizing
   The text document now needs to be represented as numerical data. This is done with the use of a count vectorizer. It turns each word that remains after the removal of stopwords as a column and gives the count of appearance that word in a row as it's value

Classification:
Label encoding of the genres was done using the mapping.csv file. The normal straight forward classification wasn't possible as the problem was that of multi-label. To get around this, one-hot-encoding of the genre column was performed. Since there were 20 genres in total, we got 20 columns each representing a genre with the values being 1 or 0 depending on whether the genre appeared on that row or not.
A logistic regression model was used here. We ran the model 20 times, each time with a different genre column and received a 1 or 0 prediction accordingly. These results were then collated to get the final output.
We got an f-1 score of 0.98107 with this model.

Part 2:

Term Frequency- Inverse Document Frequency (TF-IDF)

TF-IDF is used as a weighting factor for features.
It's an efficient way to measure the significance and originality of a word in a document of words.

The idea is that the more the appearance of a word in a document, the more it's significance. Likewise, the more documents a word appears in, the lesser will it's significance be in classified between different documents.
The TF IDF score a term t in document d can be given as follows:
TF-IDF(t, d) = TF(t, d) * IDF(t)
TF: Term Frequency
IDF: Inverse Document Frequency
Where,
TF(t, d) = (Frequency of t in d) / (Total words in d)
IDF(t) = $\log_2$(Total number of documents/ Number of documents having the term t)

The addition of inverse document frequency ensures that remaining stop words or redundant words which appear a lot aren't given importance. Also, this ensures that a term original to a document is given more weight to help better classify that document.
TF-IDF was used instead of Count Vectorizer to improve the representation of the data and subsequently the accuracy of the model.
We got an f1-score of 0.98339

Part 3:
Word2vec
The basic idea as the name suggests is that a word is converted into a vector of words.
It's a two-layer neural net which takes the text corpus as it's input and outputs the vectors representing the words in the corpus.
The idea is to group the vectors of similar words together. It is able to detect context without human intervention given sufficient data. For example, it would detect relations like man to boy and woman to girl given a sufficient amount of data based on some similarity metric like cosine similarity. Thus beyond techniques like stemming and lemmatization where words with common roots can be combined, words with similar meanings can be detected in word2vec thus invariably improving the representation of our data and the accuracy of our model.
The vectors used to represent the words are called neural word embeddings. It is in a way similar to autoencoders but instead of training for the vectors of the word by reconstructing the word as in autoencoders, word2vec trains the words against those words neighbouring it in the corpus of words.
We've used a vector size of 300 in our case.
This model yielded a macro f-1of 1

Bonus:
Since the final model yielded a macro f-1 score of 1 on the test set, no further change was made to the code.