CIS 490 Machine Learning

Final Project Report

Due: Wednesday, 26th April 11:59 PM

Submission by:

Ameh Ojonukpemi

E-mail: oameh@umassd.edu

Student ID: 01876946

Roll No: 1

Isaac Pefaur

E-mail: ipefaur@umassd.edu

Student ID: 01853843

Roll No: 28

Language Identification using Text Classification and Deep Learning

Topic: Natural Language Processing (Machine Learning Pipeline)

Introduction

The topic of our project is Natural Language Processing. This was decided as both of us in our team have had prior experiences in creating/studying machine learning. The assignment is to implement the Machine Learning pipeline so to make this a useful experience, we decided to look for a field or type of machine learning that we hadn't previously tried our hands at and so NLP was decided on as both an interesting topic as well as one with many real life applications

About The Project

While looking for NLP projects, we decided to implement a network that would not be extremely difficult but at the same time wouldn't be something basic that would warrant little to no effort. Looking online and at the process of creating NLP networks, Language Detection stood out the most. We can easily challenge ourselves without inventing new technology by creating a network by simply trying more languages, changing the way we filter the data, finding new ways to encode the data, etc. and still have a meaningful learning experience

The Dataset

As for datasets, we decided to go very basic and pull data from kaggle, where various datasets consisting of text and labels can be found in different stages of processing. We focused on Kaggle datasets in order to allow us to put the majority of our effort into the network and its functionality, rather than attempting to improve or expand performance through implementing more data.

	Text	Language
0	Nature, in the broadest sense, is the natural...	English
1	"Nature" can refer to the phenomena of the phy...	English
2	The study of nature is a large, if not the onl...	English
3	Although humans are part of nature, human acti...	English
4	[1] The word nature is borrowed from the Old F...	English
...	...	...
10332	ನಿಮ್ಮ ತಪ್ಪು ಏನು ಬಂದಿದೆಯೆಂದರೆ ಆ ದಿನದಿಂದ ನಿಮಗೆ ಒ...	Kannada
10333	ನಾರ್ಸಿಸಾ ತಾನು ಮೊದಲಿಗೆ ಹೆಣಗಾಡುತ್ತಿದ್ದ ಮಾರ್ಗಗಳನ್...	Kannada
10334	ಹೇಗೆ ' ನಾರ್ಸಿಸಿಸಮ್ ಈಗ ಮರಿಯನ್ ಅವರಿಗೆ ಸಂಭವಿಸಿದ ಎ...	Kannada
10335	ಅವಳು ಈಗ ಹೆಚ್ಚು ಚಿನ್ನದ ಬ್ರೆಡ್ ಬಯಸುವುದಿಲ್ಲ ಎಂದು ...	Kannada
10336	ಟೆರ್ರಿ ನೀವು ನಿಜವಾಗಿಯೂ ಆ ದೇವದೂತನಂತೆ ಸ್ವಲ್ಪ ಕಾಣು...	Kannada

10337 rows × 2 columns

The dataset, visualized above with Pandas, has 10,000 + entries. The Text also includes noise and undesirable characters and symbols, which had to be addressed in the preprocessing stages.

Methodology

Our process includes 3 main steps:

Preprocessing and Data Preparation

During this phase, we focused on gathering all the data we could, and making sure we implemented a universal method of processing data. This allowed us to streamline the preprocessing and increase the speed of new predictions. It also served to facilitate easier re-training with new data for in the future when we want to expand its functionality.

- We first removed all unnecessary characters, i.e. symbols.
- Converted all characters to lowercase
- Tokenize the sentences, constructing numeric representation
- The next step here is up in the air for now, but depending on our final implementation, we first need to convert our data into vectors, with each dimension representing a word. Additionally, a couple of libraries could do that with the tokens inside the network themselves, i.e. using TensorFlow's Embedding Layer as the input layer.
- Finally, the data will be split into train and test; for now we will aim for an 80-20 split

Model Selection and Implementation

This is a big part of the Machine Learning Pipeline process and so to properly reflect a thorough study and consideration we decided to test out two models and find out which is better suited or has the better process.

We initially used the SKLearn MultinomialNB model. This was supported by a vectorised input, achieved using SKLearns CountVectorization function.

```
[ ]  from sklearn.naive_bayes import MultinomialNB

     model = MultinomialNB()
     model.fit(x_train, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

Next, we employed the use of a fairly more transparent model, and decided to create a tensorflow model. To support this, our input format changed and we utilized arrays of tokenized words, using Tensorflow's Tokenizer function. The structure of the model is as follows:

```
[ ]  model_em = tf.keras.Sequential([
         tf.keras.layers.Embedding(50000, 20, input_length=sent_len),
         tf.keras.layers.GlobalAveragePooling1D(),
         tf.keras.layers.Dense(24, activation = 'relu'),
         tf.keras.layers.Dense(17, activation = 'softmax')
     ])
     model_em.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

We decided to go with the second model as it afforded us more insight into what the algorithm actually looked like and we could more easily explore the features and different moving parts that powered this model, than with MultinomialNB, which we couldn't really take a deeper dive into.

Evaluation, Prediction/Inference, and Results

In addition to outputting the final accuracy, loss, recall, etc. for both training and testing data, we can also analyze some visualizations, and included is a confusion matrix to visualize not only what the final predictions are but in general which languages the model gets wrong and if its answer makes sense in anyway (and if they don't what that means or how we can fix that), as a way to further develop the model:

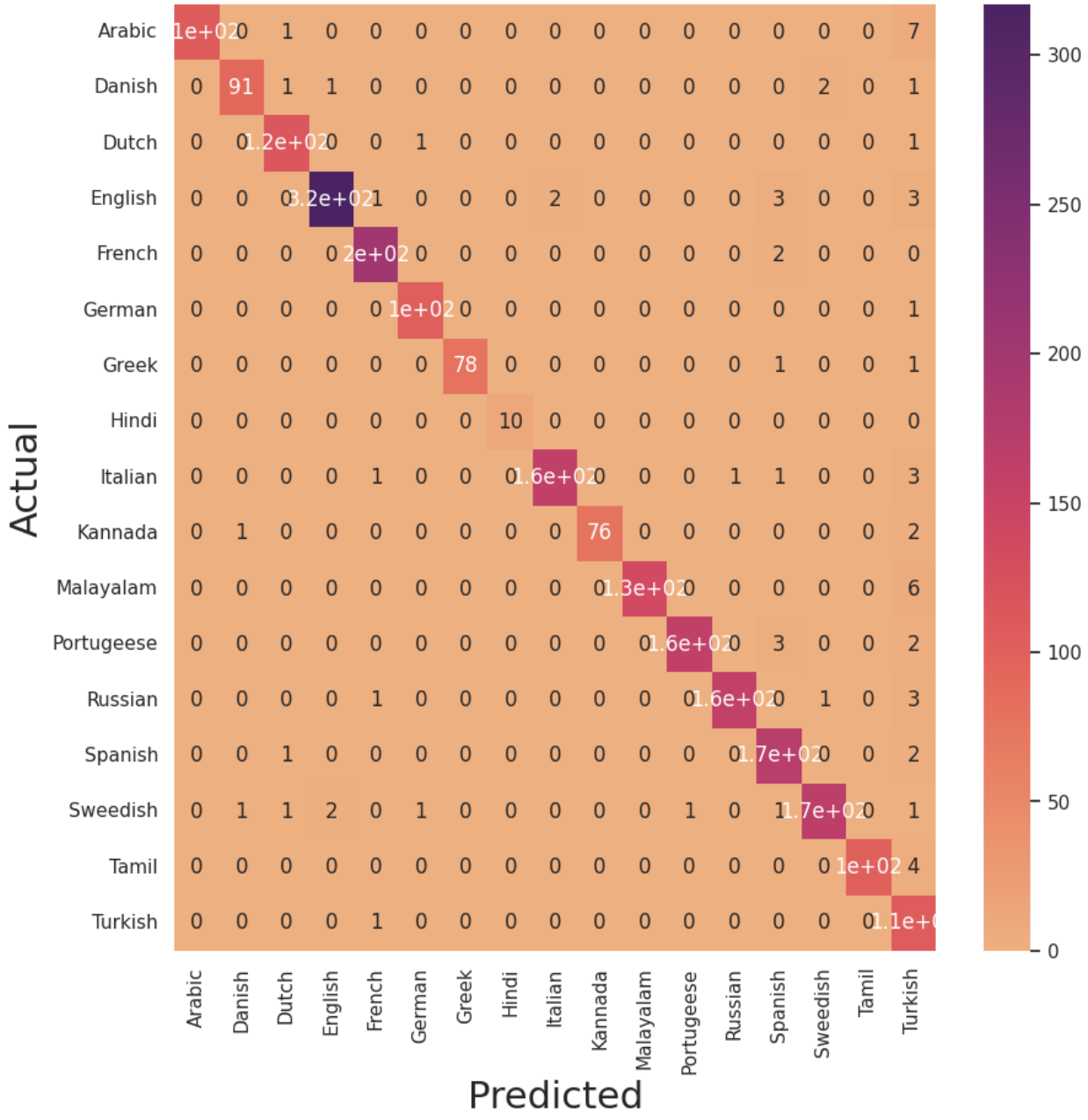
	precision	recall	f1-score	support
0	1.00	0.93	0.96	116
1	0.98	0.95	0.96	96
2	0.97	0.98	0.97	118
3	0.99	0.97	0.98	326
4	0.98	0.99	0.99	206
5	0.98	0.99	0.99	105
6	1.00	0.97	0.99	80
7	1.00	1.00	1.00	10
8	0.99	0.96	0.98	168
9	1.00	0.96	0.98	79
10	1.00	0.96	0.98	140
11	0.99	0.97	0.98	167
12	0.99	0.97	0.98	164
13	0.94	0.98	0.96	175
14	0.98	0.95	0.97	175
15	1.00	0.96	0.98	104
16	0.74	0.99	0.85	108
accuracy			0.97	2337
macro avg	0.97	0.97	0.97	2337
weighted avg	0.97	0.97	0.97	2337

	precision	recall	f1-score	support
0	1.00	0.95	0.98	104
1	0.98	0.93	0.95	90
2	1.00	0.97	0.98	89
3	0.88	1.00	0.93	261
4	0.98	0.98	0.98	191
5	1.00	0.99	0.99	95
6	1.00	0.97	0.99	72
7	1.00	0.93	0.97	15
8	0.99	0.96	0.97	158
9	1.00	0.97	0.99	71
10	1.00	0.99	1.00	129
11	0.99	0.98	0.98	124
12	0.99	0.99	0.99	140
13	0.98	0.98	0.98	193
14	0.97	0.97	0.97	151
15	1.00	1.00	1.00	83
16	1.00	0.90	0.95	102
accuracy			0.97	2068
macro avg	0.99	0.97	0.98	2068
weighted avg	0.98	0.97	0.97	2068

Above to the right we have the scores for our first model. It performed extremely well in most categories (**avg. of 98%**) and we can see that there is a generally impressive score of precision especially (**99%**). This also lends to the fact that the preprocessing done here was extremely useful as the vectorised input seemed to allow for a simpler model structure and yet produce such excellent results. On the left are the scores from the second model. It has an ever so slightly lower score than our first model (**avg of 97%**) but we still decided to go with this model for a couple of reasons:

- The preprocessing for this model is much simpler to follow and in a situation where the original functions that carried our preprocessing were not available, the structure could be replicated in plain Python even if need be.
- The second model offers more insight into what is happening behind the scenes, as mentioned earlier. Additionally, difference in accuracy is not very large so we went with one that would allow us to demonstrate the model and be able to explain in enough detail what is happening

Below, we also included a confusion matrix of our second model. You can immediately see how accurate it is. The numbers are a bit hard to read but the important part is the very low false positives in basically all categories. This proves that our method is an effective way to recognize languages without any mixup. You can especially note that even languages like English and Daish which are notoriously similar and use the same letters do not trip up our model as well. This could be attributed to the way tokenisation is done based on words not letters,this was a major influence in deciding that as languages like these could have some of the same letters but not the same words:



Tools

Our primary coding language is Python, in particular we are employing the use of IPython notebooks to facilitate easy code sharing and the convenience of easily importing most ML libraries. To create the project the necessary libraries needed include:

- Pandas (preprocessing)
- Numpy (preprocessing)

- re (preprocessing)
- Tensorflow (model building and preprocessing)
- SKLearn (model building and preprocessing)
- Seaborn (visualization)
- matplotlib (visualization)

References

Natural Language Processing - Tokenization (NLP Zero to Hero - Part 1)

Tensorflow documentation

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

SKLearn documentation

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

Code

Initial attempt with SKLearn

https://colab.research.google.com/drive/1iBd6xlZbN93tik5AKJQcqUfo0tzaz6DW?usp=sharing

Attempt two with Tensorflow

https://colab.research.google.com/drive/1Udyze6FTHB1PRZXTmQLkc9VUCpo77F-J?usp=sharing