Relation Extraction from Natural Language

Steven Au

Abstract

The testing of several machine learning approaches on text relations for NLP 243 Assignment 1.

1 Introduction

The project revolves around textual classification, specifically linking movie-related utterances to corresponding relations. This is supervised learning: we design features and inputs to predict labels correctly. We've got guidelines on which models to use, be it traditional machine learning or neural networks. Our primary objective is achieving top-notch accuracy. We'll be testing our model on a dataset with concealed true labels, submitting our predictions to Kaggle to gauge accuracy. To optimize our results, we'll delve into various machine learning strategies, enhance our feature engineering, preprocess data, analyze patterns, and fine-tune parameters.

1.1 Dataset Overview

The dataset comprises user utterances and their relations. It's a multi-label classification challenge, meaning an utterance can map to multiple labels. These utterances are in lowercase, devoid of punctuation, and can be as short as a word or as long as a sentence. The labels, termed 'core relations', follow a specific format: a primary prefix followed by a sub-label, separated by a period and underscore respectively. Prep the utterances and core relations for effective model training.

Dataset Statistics

- Total number of rows: 981
- Length of utterances:
  - Average length: Approximately 34.71 characters
  - Minimum length: 5 characters
  - Maximum length: 107 characters
- Number of repeated utterances: 11
- Number of unique individual labels: 19
- Number of unique label combinations: 47

2 Models

The models will be divided into chunks based on when I completed them and gained more understanding of the task at hand. These first models, where a base model to make sure I was able to train a simple machine-learning approach with simple embedding methods and data splits for logistic regression. These second models are neural network approach looking at layer architecture, pre-trained features, and hyper parameters. These third models consists of data preprocessing to create a hybrid-embedding approach on models that will capture more of text classification using algorithm solvers.

2.1 First Models

2.1.1 Preliminary Research

Since this is my first independent assignment to work on involving machine learning, my goal was to research my known methods and formulate a plan. With some basic NLP and ML knowledge, I started with a basic logistic regression model. The first challenge was best-representing utterances and core relations into a vector that can be trained into quantifiable results. I wanted to do some binary encoding for the core relations labels but was not sure how to start. For the utterances, I thought word2vec would be the best tokenizer technique to create semantic relations. However, knowing the concepts was a lot different from implementing them. I began researching one-hot coding the core relation column. I tried to manually code the columns by splitting the column rows and returning the unique labels into an array. I was stuck trying

ID	UTTERANCES	CORE RELATIONS
0	who plays luke on star wars new hope	movie.starring.actor movie.starring.character
1	show credits for the godfather	movie.starring.actor
3	find the female actress from the movie she's the man	movie.starring.actor actor.gender
159	how old is steven spielberg	person.date_of_birth
165	how much did it cost to produce the dark night	movie.estimated_budget

Table 1: Sample dataset table.

to manipulate the pandas data frame to create the desired results. My initial searches proved fruitless as one-hot encoding was not the right approach, and discovered the scikit library for MultiLabelBinalizer through several stack overflow posts. I now understood my problem was a text classification on a multi-class label binary encoder.

2.2 Model 1 : Logistic Regression

I used a Binary Multi-Class Label to encode the core relations that were split by spaces and simple word embeddings of Bag of Words (BoW) and Term-Frequency Inverse Document Frequency (TF-IDF). I picked straightforward embeddings for my base model.

2.3 Second Models

2.3.1 Additional Research

After a week of learning the various techniques to train a neural net and complimentary starter code for reference, I had a better idea of starting a new model. A neural net will allow neurons to capture specific information and feed it forward for better convergence and accuracy. I was hoping to get better results by implementing a more sophisticated model. I wanted to try different word embeddings on top of my previous embeddings, like n-grams, sentence transformers, and word2vec. I also wanted to try another traditional method of K Nearest Neighbors (KNN) thinking that the text classification was a clustering problem and could capture non-linear data as opposed to logistic regression.

2.4 Logistic Regression (LR)

All models use a multi-label binary encoding on the individual core relation labels. The model tests Bag of Words (BoW) and (TF-IDF) on the raw utterance strings while testing the C regularization parameter.

2.5 Multi-Layer Perceptron (MLP)

Testing 3 new word embeddings of N-gramm, random weight vector, and sentence transformer embedding with sequential neural network with 2 hidden layers. The best model implements a bootstrap oversampling method.

2.6 K-Nearest Neighbors (KNN)

Testing the best neural network model on a K-nearest neighbors model to capture any potential clustering in the data.

2.7 Third Models: More data techniques

After having time to evaluate my previous model implementations, I wanted to do some pre-processing to reduce noise in the utterance data and hope that Support Vector Machines can do both well in single-label prediction as well as multi-level predictions. My goal was to eliminate movie titles, remove common words, and remove infrequent words. I want to see how my label predictions were wrong and change how I processed the data to improve the accuracy of certain labels. I was also figuring out ways to do iterative or stratified splits to allow labels to be seen across the data splits.

2.8 Support Vector Machines (SVM)

Separate BoW utterance embedding with various data cleanings and oversampling of SVM model with a small Grid search algorithm,

2.9 Random Forest (RF)

Separate BoW embedding with keyword-based weight adjustment on SF model with Random Grid search.

3 Experiments

3.1 Basic Metrics

- Step loss and accuracy:
  Ex. `loss: 0.0047 - accuracy: 0.9438 - val_loss: 0.0736 - val_accuracy: 0.8`
- Classification Report: Refer to Table 5
- Metric scores for data split: Refer to Table 4
- Kaggle test accuracy: Refer to Table 2
- Mislabeled Utterances: Refer to Table 7

3.2 Logistic Regression

3.2.1 Training Configurations

Regularization: penalty='l2' **Parameters**: C = 3.0 **Max Iterations**: 100 **Data Split**

- **Training set (80%)**: Used to train the model.
- **Test set (20%)**: Used to evaluate the model's performance on unseen data.
- **Cross Validation**: cv=3

Model Architecture

- **MultiOutputClassifier**: To wrap the the logistic regression classifier.

3.3 Model 2: NN Configurations

3.3.1 Training Configurations

- **Loss function**: Binary Crossentropy.
- **Optimizer**: Adam optimizer with a learning rate of 0.0005. Gradient values are clipped to a maximum absolute value of 1.0.
- **Batch Size**: 32.
- **Epochs**: Up to 50 with early stopping.
- **Early Stopping**: Monitors the validation loss.

Data Split: The dataset is split into three parts:

- **Training set (60%)**: Used to train the model.
- **Validation set (20%)**: Used for hyperparameter tuning and early stopping.
- **Test set (20%)**: Used to evaluate the model's performance on unseen data.

Input Layer:

- 512 neurons.
- Uses the **He Normal** initialization method.
- Employs **L2 regularization** with a strength of 0.01.
- Activation function: **ReLU**.
- **Dropout** of 30%.

Hidden Layer 1:

- 256 neurons.
- Similar configuration to the input layer.

Hidden Layer 2:

- 128 neurons.
- Similar configuration to previous layers.

Output Layer:

- Number of neurons corresponds to the number of labels.
- Activation function: **Sigmoid**.

3.4 KNN Configurations

3.4.1 Training Configurations

- **n_neighbors**: 25, 50, 100
- **Parameters**: 'uniform', 'distance'
- **n_jobs**: -1, 1

Data Split

- **Training set (80%)**: Used to train the model.
- **Test set (20%)**: Used to evaluate the model's performance on unseen data.

Model Architecture

- **MiniBatchKmeans** n_clusters = 50
- **KNeighborsClassifier** `n_neighbors=10, weights='distance', algorithm='auto', p=2, metric='euclidean', n_jobs=-1`
- **MultiOutputClassifier**: To wrap the the logistic regression classifier.

3.5 SVM Configurations

3.5.1 Training Configurations

Random Grid Search CV

- `n_estimators: Number of trees in the forest` - 10, 25, 50,
- `max_features: The number of features to consider when looking for the best split` - 'auto', 'sqrt', 'log2'
- `max_depth`: 10, 20, 30, 40, 50, None
- `min_samples_split`: 2, 5, 10
- `min_samples_leaf`: 1, 2, 4

Data Split

- **Training set (80%)**: Used to train the model.
- **Test set (20%)**: Used to evaluate the model's performance on unseen data

4 Results

My motivation was to conduct more data analysis as my scores were not improving and faced a high variance with the Kaggle test. I mainly tried different methods. With my first models, I created a baseline

4.1 Model 1

For my first model, I was looking for a high raw accuracy on my logistic regression on the test set. At the time I was not sure how it would perform on the Kaggle set. I was mostly comparing the word embedding methods of BoW and TF-IDF with the C parameter while trying different data splits: 80/20, 60/20/20, and 80/10/10. I was able to reduce overfitting for logistic regression by 4% on Kaggle.

4.2 Model 2

On my second models, I was paying attention to the learning rate and observing how fast the convergence was for the NN model. I was excited to see a high training accuracy of over 90%, but soon realized that my test score was significantly lower, around 70%. My model exhibited high variance and was overfitting the test data. I decided to monitor my epoch scores at each step, specifically observing `val-loss` and `test-loss`. My goal was to achieve a high test accuracy with minimal discrepancies between training and testing. I experimented with various parameters: such as ReLU, dropout, learning rate, and early stop. The neural network showed better consistency between train and test accuracy since I added a regularization function, but its performance on Kaggle was comparable to that of my logistic regression model. KNN performed poorly with n-grams and did way better with sentence transformers in training. N-gram did not get above 40% and was abandoned after some configurations, while the sentence transformer got 69$ test accuracy.

My main focus then shifted towards finding a better representation for word embeddings. My best model employed pretrained *sentence-transformers*, which excelled at capturing semantic relations in contrast to the BoW n-grams. This model was the first to surpass my logistic regression score on Kaggle, improving accuracy by 2%.

4.3 Model 3

I did an iterative approach of trying different data-processing and running the model again with different parameters to see any improvement in test accuracy. I tried creating a simple rule to extract movie titles as a lot of the utterances had the same beginning keywords like "of", "the", "movie", "from", "in", "for". It did pretty well but not so well for the low-frequency utterances as those sentence structures were a lot different. I scrapped the keyword approach and analyzed the BoW corpus to see how I could remove uncommon words.

This proved to be helpful for high-frequency labels and not for low labels. I tried a new embedding of creating a separate BoW for each label, hoping that each BoW can accurately represent each corpus. This way I can delete different thresholds of uncommon words for each BoW corpus. The problem was concatenating the matrix for training and making the linear SVM training longer. This process was tedious and time-consuming as I was changing the threshold for `min_df` for count tokenizer and editing common words. The results were mostly like my logistic regression score. I decided to stop training the SVM model and attempt a random forest model.

Accuracy	**LR**	**NN**	**KNN**	**SVM**	**RF**
Test	.62	.89	.69	.69	.77
Kaggle	.60	.67	.50	.64	.55

Table 2: Best Accuracy scores for different models on Test and Kaggle datasets.

4.4 Oversampling

Since my Kaggle accuracy was substantially lower than my test accuracy, I am able to conclude the Kaggle set had more labels where the model wasn't trained. From the classification report, there were labels that had no support. When I did bootstrap oversampling by sampling the same utterance to be in a stratified data split, I improved my Kaggle score from 60% to 67%. The issue with oversampling is that it can create high bias depending on how much it is sampled. I tried to create a threshold for what labels were sampled up to 30 labels since that was the median of high-frequency and low-frequency labels. From various oversampling techniques, the mode accuracy can improved if done properly (1) as noted in my case.

5 Conclusion

I took a more abroad approach of trying various machine learning models and approaches. This was a lot more time-consuming and gave me a lot more knowledge. However, it was not the best approach for improving my score. I gained a lot of valuable skills and learned the importance of having a balanced data set and data is not always represented in the "real world" data. Overall, I learned how to apply a variety of machine learning models with their pros and cons, and the process of data engineering.

6 References

References

References

[1] Kong J, Rios T, Kowalczyk W, Menzel S, Bäck T. *On the Performance of Oversampling Techniques for Class Imbalance Problems*. Advances in Knowledge Discovery and Data Mining. 2020 Apr 17;12085:84–96. doi: 10.1007/978-3-030-47436-2_7. PMCID: PMC7206329.

A Appendix

Here, include more detailed tables, plots, figures, or any supplementary material.

Table 3: SVM separate TF-IDF Oversample

Data Type	**Accuracy**	**F1 Score**
Training Data	0.5343	0.7243
Validation Data	0.4640	0.6507
Test Data	0.4742	0.6701

Table 4: NN BoW

Metric	**Value**
Accuracy	0.6199
Precision	0.9602
Recall	0.6280
F1 Score	0.7594
Hamming Loss	0.0226

Table 5: NN ST with Oversample Classification Report

Label	**Precision**	**Recall**	**F1-Score**	**Support**
actor.gender	0.00	0.00	0.00	1
gr.amount	1.00	0.50	0.67	2
movie.country	0.97	0.87	0.92	45
movie.directed_by	0.92	0.87	0.89	63
movie.estimated_budget	1.00	0.91	0.95	23
movie.genre	0.94	0.74	0.83	23
movie.gross_revenue	0.78	0.88	0.82	8
movie.initial_release_date	0.93	0.93	0.93	41
movie.language	1.00	0.95	0.97	40
movie.locations	0.00	0.00	0.00	1
movie.music	0.00	0.00	0.00	1
movie.produced_by	0.90	0.85	0.88	33
movie.production_companies	0.77	0.85	0.81	20
movie.rating	0.95	0.97	0.96	40
movie.starring.actor	0.93	0.92	0.93	62
movie.starring.character	1.00	0.33	0.50	3
movie.subjects	0.96	1.00	0.98	23
none	0.92	0.86	0.89	66
person.date_of_birth	0.00	0.00	0.00	1
micro avg	0.93	0.88	0.91	496
macro avg	0.74	0.65	0.68	496
weighted avg	0.93	0.88	0.90	496
samples avg	0.89	0.89	0.89	496

Table 6: Frequencies of Individual Labels

Label	**Frequency**
movie.starring.actor	355
movie.directed_by	346
none	319
movie.country	223
movie.language	220
movie.rating	211
movie.initial_release_date	189
movie.produced_by	175
movie.genre	129
movie.subjects	94
movie.production_companies	93
movie.estimated_budget	79
movie.gross_revenue	34
movie.starring.character	22
actor.gender	9
person.date_of_birth	6
gr.amount	5
movie.music	3
movie.locations	3

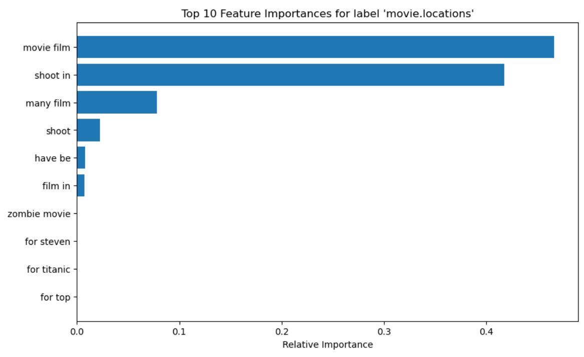

Figure 1: BoW Top 10

Table 7: Rows that were predicted wrong

ID	**Original Text**	**True**	**Predicted**
1586	find danish movies	movie.country, movie.language	
620	how do they speak in the movie the professional	movie.language	
932	show me scifi fantasy movies	movie.genre	
2282	i want to watch a drama made in china	movie.country, movie.genre	movie.country

Table 8: Frequencies of Label Combinations

Label Combination	**Frequency**
movie.directed_by	323
none	319
movie.starring.actor	302
movie.rating	189
movie.produced_by	172
movie.initial_release_date	142
movie.language	139
movie.country	136
movie.genre	101
movie.subjects	90
movie.production_companies	84
movie.estimated_budget	78
movie.country, movie.language	72
movie.gross_revenue	28
movie.starring.actor, movie.starring.character	19
movie.initial_release_date, movie.starring.actor	18
movie.initial_release_date, movie.rating	12
actor.gender, movie.starring.actor	9
movie.directed_by, movie.initial_release_date	9
person.date_of_birth	6
movie.directed_by, movie.starring.actor	6
movie.initial_release_date, movie.production_companies	6
movie.country, movie.genre	6
movie.country, movie.genre, movie.language	5
gr.amount, movie.gross_revenue	5
movie.genre, movie.rating	4
movie.starring.character	3
movie.music	3
movie.directed_by, movie.genre	3
movie.genre, movie.language	3
movie.genre, movie.subjects	3
movie.country, movie.rating	2
movie.directed_by, movie.estimated_budget	1
movie.locations	1
movie.directed_by, movie.locations	1
movie.directed_by, movie.produced_by	1
movie.directed_by, movie.subjects	1
movie.genre, movie.production_companies	1
movie.genre, movie.gross_revenue	1
movie.country, movie.genre, movie.language, movie.rating	1
movie.country, movie.locations	1
movie.directed_by, movie.rating	1
movie.rating, movie.starring.actor	1
movie.production_companies, movie.rating	1
movie.initial_release_date, movie.produced_by	1
movie.genre, movie.initial_release_date	1
movie.produced_by, movie.production_companies	1