Ben Scuron

Professor Wenpeng Yin

Foundations of Machine Learning

6 December 2022

Paraphrase Identification

Table of Contents

1. Project Description

The goal of this project was to build a multi-layer perceptron (MLP) model to classify whether two sentences express the same meaning or not. In other words, given two sentences the model should be able to predict whether the sentences are paraphrases of each other. A paraphrase is a rewording of something written or spoken by someone else. Take the sentence: "It is cold outside". A paraphrase of the sentence could be: "It is chilly outside". If trained on similar data, the MLP model should be able to classify the two sentences as having the same meaning.

2. Model

The model used in this project is the fully connected multi-layer perceptron (MLP) model. A multi-layer perceptron is a series of perceptrons interconnected between multiple layers. The first layer in an MLP model is called the input layer. The input layer is where the features of the data being predicted is fed into. The output layer is where predictions are outputted. In the output layer, there are two neurons, one for true (the sentences are paraphrases), one for false (the sentences are not paraphrases). Each layer of the MLP is a series of inter-connected perceptrons. To explain simply, a perceptron is the inputs times the weights plus the bias passed into an activation function to introduce non-linearity. In my MLP model, there are six neurons in the input layer, because I defined six features that best describe the dataset. Each input in the first layer is connected to each neuron in the hidden layer. Each neuron in the hidden layer accepts as input, output from the previous layer (the input layer). The inputs are multiplied by the weights and summed with the bias then passed into an activation function. The activation function that I used in my MLP model is the Rectified Linear Unit (ReLU) activation function. The ReLU activation function is defined as max(0, x). Meaning, if the output of a neuron is greater than 0, it will be used as is as input to the next layer. If the output of a neuron is less than 0, it will be clamped to the value 0.

The specific model that I used in this project is the MLPClassifier model from scikit-learn. The model is known to be easier to use than PyTorch models, but it is far less customizable. My model used one hidden layer consisting of 100 neurons. I tried using GridSearchCV from scikit-learn to try different combinations of layers and layer sizes but found that one layer had the best performance. The activation function used for the hidden layer is the ReLU function described above. The solver used for weight optimization is the ADAM stochastic optimizer. If you would like to learn more about the ADAM optimizer, please see this paper here: https://arxiv.org/pdf/1412.6980.pdf. ADAM is known to work well on datasets with over a thousand samples or more. The batch size of my model is 200. This is the size of the samples evaluated before updating the training loss. The initial value of the learning rate is initialized to 0.001.

3. Data

The three sets of data used for this project are: the training set, the development set, and the testing set. The training set is used to train the MLP and adjust the weights of the model to make predictions. The development set is used to validate the training set; therefore the development set is also known as the validation set. The testing set is what we want to predict. Both the training set and the development set are given the gold truth labels, while the testing set is not. The training set that I used has 7801 samples. Each sample contains two sentences, and a boolean whether the sentences are paraphrases. The development and testing set both have 4000 samples containing 1000 positive samples and 3000 negative samples. As you can see, the data in each set is not balanced (there are more negative samples than positive). To fix this, I used an Synthetic Minority Over-sampling Technique known as SMOTE from imbalanced-learn. SMOTE adds samples to each dataset to balance the data. This step is very important because the MLP model expects the data to be balanced.

4. Data Preprocessing

The data preprocessing step stayed the same as from the midterm project. I used lemmatization and punctuation removal to preprocess the data. Lemmatization is the process of converting every word to to its base form of all its inflectional forms.

5. Features

I used 6 features to feed into the MLP model. The features I used were: Levenshtein distance, cosine similarity, length difference between sentences, the number of shared words between sentences, the number of shared words between sentences, and the NIST score. These are the same features that I used in the midterm project, except that I removed the BLEU score from the model. For this reason, the explanation of each feature has stayed the same.

Levenshtein distance is a metric to determine the difference between sentences. Levenshtein distance calculates the number of edits a sentence would need to make to be equal to the other sentence. For this reason, Levenshtein distance is also known as edit distance.

Cosine similarity is another metric that is used to compare sentence similarity. Cosine similarity measures the similarity between two vectors of an inner product space. Since cosine similarity operates on vectors, each sentence must be converted to a vector. To complete this task, I converted each sentence to its TF-IDF vector representation. The corpus used was each sentence in the training data set.

The number of shared words was calculated by keeping a count of how many times each word appears in each sentence. There were two dictionaries to keep track of the count. The first dictionary would store the counts of words from sentence 1, and the second dictionary would store the counts of words from sentence 2. After the words were counted, the common words from each sentence were collected into a set, and the minimum count for each word in the dictionary was added to a list. In this way, a sentence: "I love data science" and "I hate data science" would have a shared words count of three words: "I", "love", "data science".

The next feature that was created was the number of shared parts of speech between sentences. This feature was implemented using the part of speech tagger from the nltk library. A similar technique to the number of shared words was used to keep track of the same number of parts of speech.

The last feature implemented was the NIST score. The NIST use n-gram co-occurrence statistics to evaluate the quality of sentence translation. While not traditionally used for this task, I found that using this score improved my development accuracy substantially.

There are other features that I implemented that I decided to not include in the final model. The reason why I did not include all the implemented features is because they did not have a good effect on dev accuracy. Examples of extra features were: 'ratio', 'partial ratio', 'token sort ratio', 'Ratcliff Obershelp score', 'BLEU score', and 'Jaccard similarity'

6. Algorithms and Libraries

I used a variety of libraries throughout the implementation of this MLP model. The libraries I used are pandas, numpy, ssl, nltk, string, Levenshtein, sklearn, and difflib. The algorithms that were used in this project were mostly abstracted away in the form of imported library functions. The MLP model was used to make predictions using the features discussed in the last section. The same libraries used in this project were also used in the midterm project. However, some of the methods provided by these libraries were not used in the midterm project. For instance, to tune my MLPClassifier model, I used GridSearchCV from scikit-learn. This method was not used in the midterm but was used in this project.

7. Experience

Throughout the course of this project, I learned many new skills. The first skill that I learned was how to use existing code to create a better model. For instance, I used my features from the midterm project and utilized them into a completely different model that performs very well. I was able to achieve ~90% accuracy and ~82% F1 score on the development set. Another skilled that I acquired through this project was the ability to tune an MLP model. Before this project, I had no clue how to tune a model. From my understanding now, a lot of the tuning portion is a trial and error. Luckily, I found a function from scikit-learn that can help with this portion of the problem (GridSearchCV). Another important skill that I learned was how to balance data via oversampling. Using SMOTE from imbalanced learn made it very easy to balance each of the data sets. Balanced data is very important when working with an MLP model. While not a new skill, I furthered my understanding of data preprocessing, and feature extraction. Once again, I am glad that I started working on this project very early on, because working on machine learning is not a quick process.