10/04/2021

MIDAS@IIITD Summer Internship Task

(Task -3 [NLP])

Shashwat Jha

This task requires us to predict the product category by using its description as the main corresponding feature. The dataset (https://bit.ly/3d64wK3) consists of 20000 rows and 15 columns. Please refer to the code (https://bit.ly/3uJEimP) to understand the following explanation better. We will be using Jupyter Notebook for coding.

Loading Dataset:

We load our dataset, which is a csv (comma-separated values) file using pandas library. The code is as follows:

pandas.read_csv("name_of_file.csv")

Pandas library provides numerous methods such as *.head, .shape, .columns, .describe, .info* etc. which can be called to explore our dataset before we proceed to data preprocessing part.

Data Preprocessing:

As we have mentioned earlier that our main feature for model training will be product "*description*", we will see what preprocessing steps are needed before we can feed this data for training. Each product has a description which is of string type. We can notice that there are some characters in these columns which are not required as they are non-important, for example hyphen, special characters, whitespaces etc. To get rid of these characters we will use the regex library to clean strings. We also check for any missing descriptions. In our dataset there are only 2 products overall, which doesn't have a description. So, we decide to remove these two rows from the observation.

Once we have cleaned the strings the next step is to take a look at our target column i.e., "*product_category_tree*". As we explore this, we notice that there are many products which does not have a definite product category i.e., instead of categorizing a product under say clothing it is categorized with something like product detail or something which isn't "generic" (by generic I mean categories such as clothing, jewelry, automotive, electronics etc. with a more specific definition). To overcome this anomaly, we will extract only main categories. For processing these categories, we will use the *.split* method of python along with *.strip* method to clean these category tree and extract the main category.

Once we have processed our feature i.e., description and the label i.e., category we now create a data frame with only these two columns. As we know that this is a multiclass classification problem, we will now encode these classes with a numeric value for the ease of data handling.

Model Building:

Before we start with model training, we need to specify the training and testing dataset, for this purpose we will use *train_test_split* method of scikit learn. We have decided to keep the training size as 80% and testing size as 20%. We need to be careful when it comes to choosing training and testing size in order to prevent overfitting or underfitting of our model.

Once we have our training dataset and testing dataset the next step is of vectorization. As we know that our input features aren't numerical. It's not efficient to use strings as a whole to feed the model for training, to overcome this we use the method of vectorization which lets us break these strings into vector of real numbers and then use it for training purpose. (More on vectorization -https://bit.ly/3mBF6Hh). There are different types of vectorization methods, but in this problem, we will restrict ourselves to only two of them which are CountVectorizer and TfidfVectorizer.

CountVectorizer - It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

TfidfVectorizer - TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

We can refer to this video for visualizing the difference between CountVectorizer and TfidfVectorizer: https://bit.ly/2QgYvBd

Once we have vectorized our training and testing dataset the next step is to choose the best possible algorithm for our model.

We will also use Pipelines and transformers to ease our task of model training.

For this classification we are going to use the following algorithms:

- Multinomial naïve bayes
- Support Vector Machine
- Logistic regression
- Stochastic Gradient Descent

We can read this article and understand which all algorithms can be used for text classification- https://bit.ly/3dO6Qoo

The next step is to fit the model with our training feature and training label for each of the algorithms.

Model Evaluation:

The reported accuracy of the different algorithms used on our testing data is as following:

- Multinomial naïve bayes
  - With CountVectorizer – 93.59%
  - With TfidfVectorizer – 87.21%

- Support vector machine – 97.50% (best)
- Logistic Regression – 96.33%
- Stochastic Gradient Descent – 94.86%

We can see that most of the algorithms we used gives an accuracy of over 90% which can be summarized as the fact that our data is very clean and with very few anomalies.

Testing our models:

To test our models, we can choose any product description from the testing data. We can call the *inverse_transform* to get the corresponding category from the encoded value outputted by the model.