Task Submission

Innovative Tech Lab

Nishkarsh Tripathi

August 8th, 2020

Definition

Project Overview

Nowadays, due to a lot of websites and market popping up, it has become really hard to find the exact think you are looking thus losing a lot of precious time and one such domain is car service.

In this report, I will discuss how we can improve the web results using Machine learning and Natural language processing to classify the user search query in the best possible way, thus providing user with the exact website he/she is looking for.

Problem Statement

The goal is to create a text classifier that can automatically distinguish the data into categories efficiently. Here we will categorize the car service website URL on the basis of certain keywords; the tasks involved are the following:

1. Getting the data (Web Scraping)
2. Data Cleaning
3. Preprocessing
4. Feature Engineering
5. Creating a Classifier to classify the URL into several classes.
6. Summarizing results

The final model is expected to be useful for Search Engine Optimization (SEO) and better user experience.

Libraries Used

1. **Pandas** - DataFrame Manipulation
2. **Scikit-Learn** - Scaling and Applying ML models
3. **Matplotlib and Plotly** – Data Visualization
4. **KeywordProcessor** – Find keywords in the URL
5. **Requests** - Making requests to the website
6. **BeautifulSoup** – Parsing HTML data
7. **NLTK** – Text Sentiment Extraction
8. **Tensorflow and Keras** – For training neural network models

Workflow

Data Scraping and Preparation

The task here is to scrape the website URL's and we will use two most popular libraries for this purpose- requests and BeautifulSoup.

1. Use request module and request the google website with a query using https protocol and load the page into an object. We will scrape the search result links and create a links.txt file for storing those links. **Code here**.

2. Next Step is to extract the data by going through those website links which can be done using BeautifulSoup. We can perform this task for one link and then loop over all the links and store them in a file which can be used further. Also, we can also do some IP engineering to include the features of websites like relevancy of content, Website IP status (running or not)

3. We now need to classify these website links on the basis of content we have extracted so that we have labels for classification. For this, we can use text data scraped from each website and look for keywords such as:

   Consider the example of this website- https://gomechanic.in/

   We can see that on "Our Services" section we have a lot of services listed like:

   - AC Services
   - Tyres and Wheel Care
   - Cleaning and Detailing
   - Denting and Painting
   - Batteries
   - Insurance Services
   - Light and Fitments
   - Glass and Custom Services

Following will be our classes for the dataset and note that we can have multiple classes for each of the website as a website is bound to provide a lot of services.

At the end of this step we will have a dataset with schema with columns as

i. **URL** – Contains the URL of the page

ii. **IP Status –** Status of the website

iii. **Relevancy of the Content (Optional) –** A binary class (Relevant or not)

iv. **Classes –** A list of classes for each of the rows separated by commas.

v. **Text** – Text scraped from the respective website

Exploratory Visualization

In this step we will try to visualize using various tools available to us like Matplotlib, Seaborn and Plotly. This step can help us get insights on the data we have scraped like:

i. How many links are up and running?

ii. What kind of Categories do we have in the data?

iii. In which category, most of the websites lie? – WordCloud

Preprocessing

Now that we have finished data preparation and EDA, our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:

1. Begin by removing the html tags.
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

Here is an example function to clean the **Text** Column – **Code here**

The next step in preprocessing after Lemmatization and cleaning would be to generate word or sentence vectors using either Bag of words (Count Vectorizer), TFIDF or Word2Vec.

Word2Vec is state of the art so here is a simple utility script to vectorising the cleaned text – **Code here.**

Classification based on keywords

For any machine learning algorithm, we need some training set and test set for training the model and testing the accuracy of that model. Hence to create the set of data for the model, we already have the text from different websites, we will just classify them according to the keywords, and then apply the results in the next module.

The approach here is that we will have certain keywords belonging to the particular category, and

We will match those keywords with the text and find the class with maximum **Matching_value.**

- **Matching Value: (Number of keywords matched in a single website text) / (Total number of Keywords Matched)**

Now, we need to use **KeywordProcessor** to find the keyword inside the test received from the URLs**.** Firstly, we will create an instance of **KeywordProcessor** and add the keywords using **add_keyword()** and then find the percentage of **Matching_value**. **Code here.**

Modelling

Now, our next task is to apply models but before that we will split the data into train, validation and test set for sanity check that our model doesn't do overfitting. We can apply a simple train_test_split function for this and specifying the test data size into the arguments and then again applying train_test_split on train to obtain final train and Validation sets.

Scope of Models:

1. **Naïve Bayes –** Naïve Bayes is state of the art in text classification tasks. It uses conditional probability of a keyword being in a text corpus. The probability is what makes it work really well. There is still one more aspect to handle. Since the above equation involves probabilities of each word of a new sentence with respect to a class, if a word from the new sentence does not occur in the class within the training set, the equation becomes zero.

   To solve this issue we can apply Laplace Smoothing. As here we have multiple classes so we will employ a MultinomialNB.

2. **Support Vector Machines** – As our data is high dimensional so Support Vector Classifiers might work well as they employ kernel trick and can separate the data well in high dimensions. This algorithm will work slow so we might not want to try to train the whole model in a single go.

3. **Neural Networks (CNN)** – We can also use Convolutional Neural Networks and LSTM's as they work well with sequence data such as Text. These models will take a lot of time to train but might give surprising results.

   Below is an example for a CNN with 3 layers which uses accuracy as the optimization measure. This model might perform well or might not but we can experiment with the Layers to check what works and what now like adding more layers or adding more units.

```
def create_embedding_model(vocab_size, max_length):
    model=models.Sequential()
    model.add(layers.Embedding(vocab_size, 100, input_length=max_length))

    model.add(layers.Conv1D(1024, 5, activation="relu"))
    #model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling1D())

    model.add(layers.Conv1D(1024, 5, activation="relu"))
    #model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling1D())

    model.add(layers.Flatten())
    model.add(layers.Dense(512,  activation="relu"))
    dropout = Dropout(0.5)
    model.add(layers.Dense(15,  activation="softmax"))
    return model

embedding_model = create_embedding_model(vocab_size=vocab_size, max_length=max_length)
embedding_model.summary()

from keras.optimizers import SGD
#opt = SGD(lr=0.01, momentum=0.9)
embedding_model.compile(loss='categorical_crossentropy',
              optimizer= 'adam',
              metrics=['accuracy'])
```

Final Thoughts

We can train and test on validation set or apply Cross validation to find the best model accuracy. Also, we can visualize the word and sentence vectors using t-SNE to get more idea on how the data is distributed among the classes.

We can also use stars and reviews of websites into account to classify the results and rank them better to give the best user experience.