Machine Learning

Gradient Boosting
Bias Variance Tradeoff
Caret Package in R
Linear Regression in Julia
ARIMA Model for Time Series Forecasting
Naive Bayes Algorithm
Principal Component Analysis (PCA)
Mahalonobis Distance
Portfolio Optimization
Augmented Dickey Fuller Test (ADF Test)
Linear Regression in R
Cosine Similarity
Feature Selection Approaches
Gensim Tutorial (NLP)

SpaCy Text Classification – How to Train Text Classification Model in spaCy (Solved Example)?

by Shrivarsheni

Text Classification is the process categorizing texts into different groups. SpaCy makes custom text classification structured and convenient through the `textcat` *component.*

Text classification is often used in situations like segregating movie reviews, hotel reviews, news data, primary topic of the text, classifying customer

KPSS Test for Stationarity
Lemmatization Approaches
Numpy Tutorial – Part 1
Numpy Tutorial – Part 2
P-Value – Understanding from Scratch
Vector Autoregression (VAR)
Time Series Analysis in Python
Evaluation Metrics for Classification models
Topic modeling visualization
Topic Modeling with Gensim (Python)
Building chatbot with Rasa and spaCy
Detect outliers using IQR and Boxplots?
Detect outliers with z-score

classification model proves to be more accurate. This article shows you how to build a custom text classifier using the spaCy library

Training Custom Text Classification Model in spaCy. Photo by Jesse Dodds.

Contents

Introduction

Let's looks at a typical use case for Text Classification.

from Swiggy. These reviews help them to analyze the problems and improve the service. But let's take a peek into this process. There are millions of reviews filled out by customers, is it possible to manually go through each of them and see if it's an appreciation or a negative review?

Of course, No! The first step would be to classify all the reviews into positive and negative categories. Then, you can easily analyze how many people were not satisfied and why. This process of categorizing texts into different groups/labels is called Text Classification.

Text Classification can be performed in different ways. Here, we'll use the spaCy package to classify texts. spaCy is has become a very popular library for NLP and provides state-of-the-art components. For practical cases, it is mostly preferred to use a trained Custom model for classification. Let me first introduce to you what is a custom model and why we need it in the next section.

What is a custom Text Classifier model?

Let's say you have a bunch of movie reviews/customer reviews. You wish to classify each review as positive or negative. If you use the default categorizer of spaCy, the result is not likely to be great. Instead, what if you collect labeled data set

The results will be far better and accurate! You can do this by training a custom text classifier. You first train it on relevant labeled datasets and made ready for our use in similar context. It's very helpful especially in cases where the amount of data is huge.

In the next sections, I'll guide you step-by-step on how to train your text classification model in spaCy.

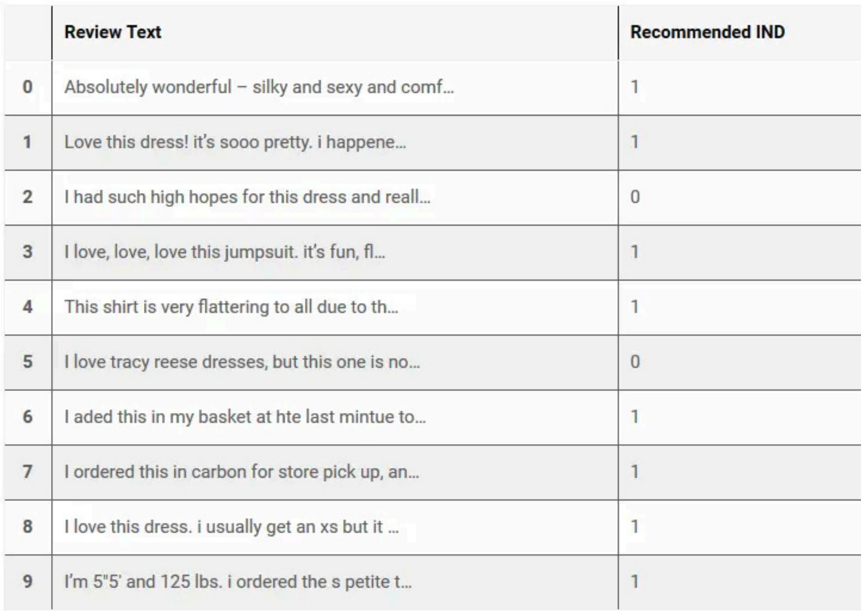
Getting started with custom text classification in spaCy

spaCy is an advanced library for performing NLP tasks like classification. One significant reason why spaCy is preferred a lot is that it allows to easily build or extend a text classification model. We shall be using this feature.

Now, I'll demonstrate how to train the text classifier using a real example. Consider you have text data containing custom reviews of cloths of a company. The task is to use this data and train our model on it. Finally, the model should be able to classify a new unseen review as positive/negative.

You can download the dataset containing reviews on clothes here. Read the data set into a CSV and view the contents. For our task, we require only 2 columns. The `" Review Text "` column containing product reviews and the `"Recommended IND "`

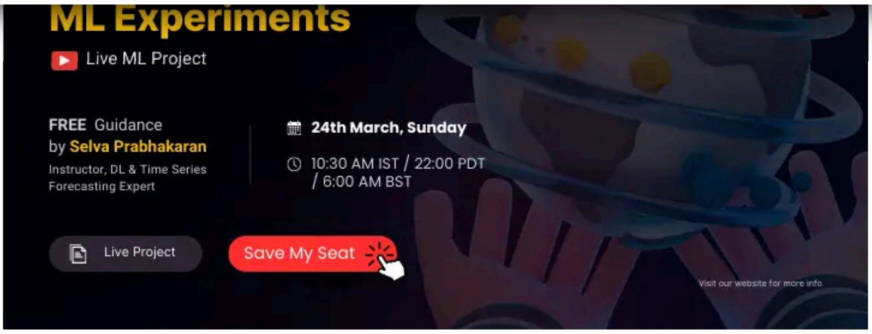
```
  # Import pandas & read csv file
       pandas    pd
reviews pd read  sv     ps   raw  i     ser  n en


reviews   reviews    eview  e      e    ended
reviews  ead
```



	Review Text	Recommended IND
0	Absolutely wonderful – silky and sexy and comf...	1
1	Love this dress! it's sooo pretty. i happene...	1
2	I had such high hopes for this dress and reall...	0
3	I love, love, love this jumpsuit. it's fun, fl...	1
4	This shirt is very flattering to all due to th...	1
5	I love tracy reese dresses, but this one is no...	0
6	I aded this in my basket at hte last mintue to...	1
7	I ordered this in carbon for store pick up, an...	1
8	I love this dress. i usually get an xs but it ...	1
9	I'm 5"5' and 125 lbs. i ordered the s petite t...	1

You can observe that for positive reviews the label is 1 and for negative reviews, it is 0. We need to train the model than can categorize new unseen reviews.

This is the raw data we have got. As we are using spaCy, to train our model let's import the package. After importing, you can load a pre-trained model like `en   re we  s`. We will be adding / modifying the text classifier to this model later. For any spaCy model, you can view the pipeline


ML Experiments
Live ML Project
FREE Guidance
by Selva Prabhakaran
Instructor, DL & Time Series Forecasting Expert
24th March, Sunday
10:30 AM IST / 22:00 PDT / 6:00 AM BST
Live Project
Save My Seat
Visit our website for more info

components present in the current pipeline through `pipe_names` method.

```
# Import spaCy ,load model
import spacy
nlp=spacy.load("en_core_web_sm")
nlp.pipe_names
```

Output:

```
['tagger', 'parser', 'ner']
```

You can see that the pipeline has tagger, parser and NER. It doesn't have a text classifier. So, let's just add the built-in `textcat` pipeline component of spaCy for text classification to our pipeline. The `add_pipe()` method can be used for this.

```
#  dd    t e    lt    te t at  ompo e t to t e p
textcat=nlp.create_pipe( "textcat", con ig= "excl
nlp.add_pipe(text_cat, last= r  )
nlp.pipe_names
```

Output:

Now, we are going to train the `textcat` with our reviews dataset using the `simple_cnn` architecture.

First, you need to add the desired labels to the pipeline component. In this case, it is `POSITIVE` and `NEGATIVE` as per the reviews. Add these labels to `textcat` using `add_label` function.

```
# Adding the labels to textcat
textcat.add_label("POSITIVE")
textcat.add_label("NEGATIVE")
```

Output:

```
1
```

How to prepare training data in the desired format?

Your default model with `textcat` is ready, you just need to prepare the data in the required format.

You can write a function `load_data()` which takes a **list of tuples as input**. Each tuple contains the text and label value(0 or 1). The below code demonstrates how to convert our reviews data set into this desired format

```
# on e ting the data  a e into a list o  t  les
re ie s['t ples']   re ie s.appl (       ro   (ro
```

Output:

Next, you can pass the data as input to a _ function.

The _ function performs the below functions:
– Shuffle the data using function
This prevents any training based on the order of examples.
– For each tuple in the input data , category is

– 80 % of the input data will be used for training, whereas 20% for evaluation. You can change this proportion using the `split` parameter.

After defining the function , pass the list of tuples `train` to the above function. The function will return you the texts and cats for both training and evaluation. You can use these to get the final training data as shown in below code.

```
        ran

    l a   ata li it    split
    train  ata train

    ran    s    l  train  ata
    t  ts  la  ls    ip  train  ata

    cats                    l


    split   int l n train  ata    split
            t  ts  split   cats  split     t  ts s

n t  ts


 train t  ts  train cats        t  ts      cats


train  ata   list  ip train t  ts     cats   cats
train  ata
```

```
[('I ordered this shirt in both white and coral.
 {'cats': {'NEGATIVE': False, 'POSITIVE': True}})
```

The above output shows you the format of the final desired training data.

How to write the evaluation function?

Till now, we have prepared the training data in the desired format and stored it in _ variable. Also, we have got the text classifier component of the model in . So, we can

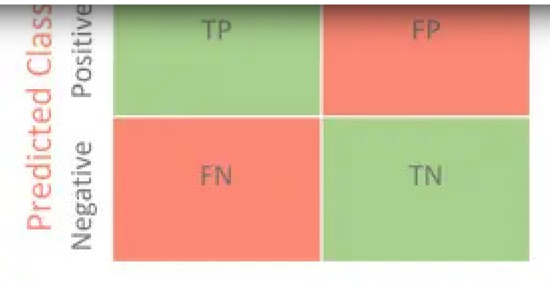
For any model we are going to train, it's important to check if it's as per our expectations. This is called evaluating the model. This is an optional step but highly recommended for better results. If you recall, the `load_data()` function split around 20% of the original data for evaluation. We'll be using this to test how good the training was.

So, let me guide you **how to write a function `evaluate()` that can perform this evaluation** process. **We will call this `evaluate()` function later during training to see the performance.**

This function will take the `textcat` , and the evaluation data as input. For each of the text in evaluation data, it reads the `score` from the predictions made. And based on this, it calculates the values of True positive, True negative, False positive and false negative.

What do they above terms mean?

- True positive (tp) : If the actual label is one 1 and prediction is also 1
- True Negative (tn) : If the actual label is one 0 and prediction is also 0
- False Positive (fp) : If the actual label is one 0 , but the prediction is 1
- False Negative (fn) : If the actual label is one 1 , but the prediction is 0



Confusion Marix

How to assign a score based on these terms?

Once you get the above values based on the predictions done by the model, you can find the evaluation metric scores. These metric scores are to tell you how good the model is.

Precision is to find what percentage of the model's positive (1's) predictions are accurate. Let's suppose you have a model with high precision, I also want to know what percentage of ALL 1's were covered. This is termed as Recall

A good model should have a good precision as well as a high recall. So ideally, I want to have a measure that combines both these aspects in one single metric – the F1 Score.

F1 Score = (2 * Precision * Recall) / (Precision + Recall)

If you want to obtain more clarity on the evaluation metric, check out various evaluation metrics for classification.

The below code writes a function `evaluate()` that can perform the above discussed evaluation when

```
    docs = (tokenizer(text) for text in texts)
```

Training the model and printing evaluation scores

The data is stored in `train_data()` variable and the component in `texcat` .

Before we train, you need to disable the other pipeline components except `textcat` . This is to prevent the other components from getting affected while training. It can be accomplished through the `disable_pipes()` method. Next, use the `begin_training()` function that will return you an optimizer.

You can also fix the no of times you want to iterate the model over the examples using `n_iter` parameter. It should be optimum. In each iteration, we **loop over the training examples and partition them into batches using spaCy's minibatch and compounding helpers**.

The `minibatch()` function of spaCy will return you the training data in batches. It takes the `size` parameter to denote the batch size. You can make use of the utility function `compounding()` to generate an infinite series of compounding values.

1. `compounding()` function takes three inputs which are `start` (the first integer value) , `stop` (the maximum value that can be generated) and finally compound. This value stored in compound is the

For each iteration , the model is **updated through the** `nlp.update()` **command.** Parameters of nlp.update() are :

1. `docs` : This expects a batch of texts as input. You can pass each batch to the zip method, which will return you batches of text and annotations.
2. `golds` : You can pass the annotations we got through the zip method here
3. `drop` : This represents the dropout rate.
4. `losses` : A dictionary to hold the losses against each pipeline component. Create an empty dictionary and pass it here.

The training is complete now. You can evaluate the predictions made by the model by calling the `evaluate()` function we defined in the previous section.

```
         spac .ut l           n  atc   co pound ng



  ot er p pes    p pe      p pe    nlp.p pe na es
       nlp.d sa le p pes( ot er p pes)
      opt    er   nlp. eg n tra n ng()


           (  ra n ng t e  odel... )
           (       t      t      t      . or at(



               range(n  ter)
```

```
        for batch in batches:
            texts, annotations = zip(*batch)
```

Output:

```
Training the model...
LOSS P R F
8.673 0.906 0.962 0.933
5.705 0.911 0.961 0.935
4.112 0.918 0.957 0.937
3.117 0.920 0.954 0.937
2.515 0.923 0.950 0.937
2.048 0.922 0.948 0.935
1.886 0.926 0.945 0.936
1.496 0.922 0.946 0.934
1.416 0.923 0.945 0.934
1.094 0.924 0.949 0.936
```

You can observe the evaluation scores. Both training and evaluation are complete!

Test the model with new examples

review of clothes. The classification or prediction of it will be stored in `doc.cats` attribute. The `Docs.cats` attribute stores a dictionary that maps a label to a score for categories applied to the document.

```
 # Testing the model
test_text="I hate this dress"
doc=nlp(test_text)
doc.cats
```

Output:

```
          I      .                    I I     .
```

You can see that the `I` score is high, classifying this as a negative review. This output is as per our expectations. yay!

Conclusion

I hope you have understood how to train a custom text classification model with spaCy. This has wide applications in every field. Similar to this, spaCy also provides an option to train your own custom NER model, you can check out the post here. We are coming up with more such juicy articles, stay tuned!