✦ Member-only story

Applying Text Classification Using Logistic Regression

A comparison between BoW and Tf-Idf

Idil Ismiguzel · Follow
Published in Analytics Vidhya
8 min read · May 8, 2020

Listen Share ••• More

Photo by Daniel Eledut on Unsplash

Creating “language-aware data products” are becoming more and more important for businesses and organizations. Leveraging on machine learning and NLP, organizations can interact with their customers both rationally and emotionally, improve their customer experience, and provide tailored assistance.

Although text is unstructured data and it is generally produced by people to be understood by other people. So how can we process a large amount of text, transform it into a representation that we can build on machine learning models to predict and classify?

If you want to get an introduction to applications of text and NLP per se, you can have a look at my previous article.

In this article, I will be investigating Amazon’s fine food reviews dataset which spans a period of more than 10 years, including ~500,000 reviews between Oct 1999 — Oct 2012. Step by step I will be building a machine learning model to determine whether a review is positive or negative.

> *You can reach all the code here in my GitHub!*

Let’s start with the basic data exploration: Dataset consists of 568,411 reviews with 9 attributes which are

1. **ProductId** unique identifier of the product
2. **UserId** unique identifier of the user
3. **ProfileName**
4. **HelpfulnessNumerator** # users who found the review helpful
5. **HelpfulnessDenominator** (# users who indicated whether they found the review helpful or not)
6. **Score** (5-star rating)
7. **Time**
8. **Summary**
9. **Text**

```
>>> df.head(2)
```

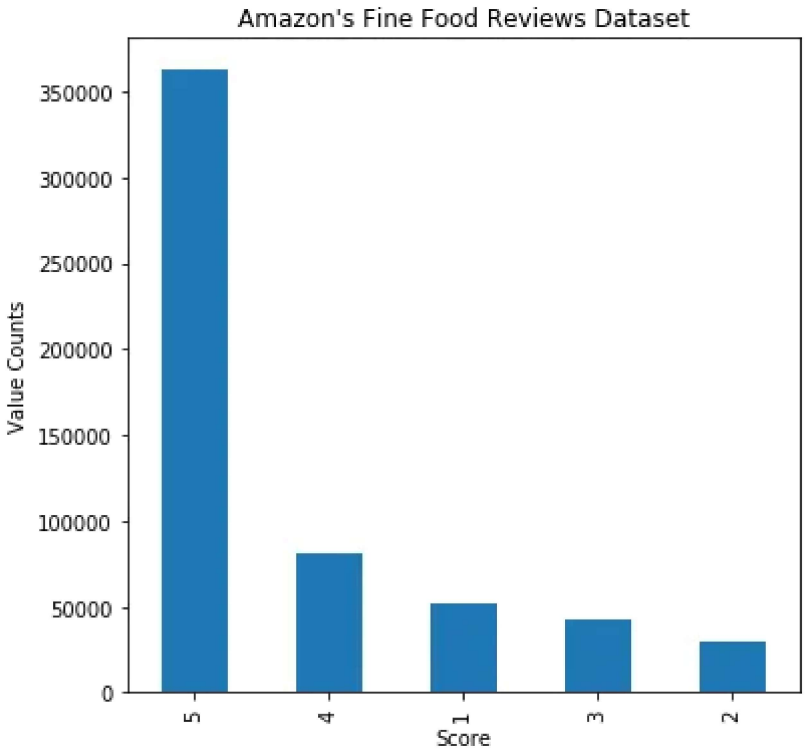
Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...

I have started cleaning the dataset by doing **data deduplication**, which means removing duplicates from the dataset. I've performed data deduplication based on a subset of "UserID", "ProfileName", "Time" and "Text" since a customer cannot post a review at the same time for two (or more) different products. Now dataset consists of 393,919 reviews.

```
>>> df.drop_duplicates(subset={'UserId', 'ProfileName', 'Time',
    'Text'}, inplace=True)
>>> df.shape

(393919, 9)
```

To understand whether the review is positive or negative, I've benefited from the 5-rating score given by the user. Later, I will be using these positive and negative labels to train supervised machine learning model and predict labels of an unseen test set.



By looking at the distribution of 5-star rating scores, it is clear that we have a lot of reviews with score 5 which leads to unbalanced classes. For this reason, I decided to treat reviews with score 4 and 5 as positive, and the rest as negative.

I've added the "Label" feature to the dataset, which equals 1 if the review is positive and equals 0 if the review is negative. 78% of the dataset has positive reviews, this highly polarized values means that even without applying any machine learning model if we have to guess whether a new review is positive it is highly likely we will be right... Now the challenge is to build a model with an accuracy higher than 78%.

Text cleaning and Preprocessing

There can be multiple ways of cleaning and preprocessing the textual data and here I have applied the ones which are frequently used in NLP pipelines.

- **Lower case:** Since we would expect to treat "Food" and "food" as the same word, without creating various predicting powers, I've down-cased each word.
- **Contractions:** I've replaced contractions with their longer forms such as "isn't": "is not", "can't": "cannot". To do so, I've imported contractions list from here.

- **Remove special characters:** I've cleaned the data from any special character such as double quotes, punctuation, and possessive pronouns.
- **Stopwords:** I've removed stopwords since they add noise without bringing any information value in modeling. I've downloaded a list of English stopwords from the nltk package and deleted them from the text corpus.
- **Tokenization:** to process text, we need to split it into smaller chunks. Here, I've split sentences into words using WordPunctTokenizer from the nltk library.
- **Lemmatization:** To convert each word into its root word, I've used Lemmatizer from WordNet.

This is the text-processor I have used to apply all the steps listed above.

```python
def clean_text(text):
    '''Text Preprocessing '''

    # Convert words to lower case
    text = text.lower()

    # Expand contractions
    if True:
        text = text.split()
        new_text = []
        for word in text:
            if word in contractions:
                new_text.append(contractions[word])
            else:
                new_text.append(word)
        text = " ".join(new_text)

    # Format words and remove unwanted characters
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\<a href', ' ', text)
    text = re.sub(r'&amp;', '', text)
    text = re.sub(r'[_"\-;%()|+&=*%.,!?:#$@\[\]/]', ' ', text)
    text = re.sub(r'<br />', ' ', text)
    text = re.sub(r'\'', ' ', text)

    # remove stopwords
    if remove_stopwords:
        text = text.split()
        stops = set(stopwords.words("english"))
        text = [w for w in text if not w in stops]
        text = " ".join(text)

    # Tokenize each word
    text =  nltk.WordPunctTokenizer().tokenize(text)

    # Lemmatize each token
    lemm = nltk.stem.WordNetLemmatizer()
    text = list(map(lambda word:list(map(lemm.lemmatize, word)), text))

    return text
```

Text-preprocessing hosted with ❤ by GitHub view raw


```
>>> df[['Score', 'Text', 'Label', 'Text_Cleaned']].sample(2)
```

Id	Score	Text	Label	Text_Cleaned
295241	5	This is my favorite Nong Shim to date. It not only tastes great but also has about half the sodium content of the others (40% vs. >90% for whole package). It's not healthy per se, but is probably a lot better for you. The dried veggies seem more substantial than the mere onion flakes in the other packs; this one has carrots, some tofu-like pieces, and green things. The sauce is also more savory than salty which is a good change of pace. Just be warned that it's korean style chajang noodles w...	1	[favorite, nong, shim, date, tastes, great, also, half, sodium, content, others, 40, vs, >, 90, whole, package, healthy, per, se, probably, lot, better, dried, veggies, seem, substantial, mere, onion, flakes, packs, one, carrots, tofu, like, pieces, green, things, sauce, also, savory, salty, good, change, pace, warned, korean, style, chajang, noodles, dark, sauce, sauce, bit, difficult, mix, evenly, tends, clump, sprinkle, evenly, noodles, mix, rather, dumping, one, spot]
357460	5	Gift for my husband...and he chuckled all day about this box of Soylent Green. Even though they are just green crackers, he won't eat them (can we really trust it?). I have had the box just sitting on my counter for the past few weeks (what do I do with a box of crackers that won't be eaten?) and a few people have noticed and broke out into gasps. That has been fun - the reaction to a box of Soylent Green in my kitchen by the unsuspecting passer-by. Well worth the money, the most talked ...	1	[gift, husband, chuckled, day, box, soylent, green, even, though, green, crackers, eat, really, trust, box, sitting, counter, past, weeks, box, crackers, eaten, people, noticed, broke, gasps, fun, reaction, box, soylent, green, kitchen, unsuspecting, passer, well, worth, money, talked, gift, holiday, season]

A sample of dataset showing the reviews before and after preprocessing (Text vs Text_Cleaned)

Feature Engineering

Since machine learning models do not accept the raw text as input data, we need to convert “Reviews” into vectors of numbers.

There are different ways of transforming text into numeric vectors. In this analysis, I’ve applied first **Bag of Words**, followed by **Bag-of-n-Grams**, and later I’ve moved to **Tf-Idf** which is a more complex representation. I’ve aimed to model two different classification by using these methodologies and compare their performances on Amazon’s dataset.

Bag of Words (BoW)

It is a simple but still very effective way of representing text. It has great success in language modeling and text classification. It is based on the word count statistics.

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> bow_converter = CountVectorizer(tokenizer=lambda doc: doc)
>>> x = bow_converter.fit_transform(df['Text_Cleaned'])

>>> words = bow_converter.get_feature_names()
>>> len(words)

110422
```

Bow(w, d)= Number of times word w appears in document d

Since bag-of-words representation converts text into a “flat vector of numbers”, it does not remember any original textual sequence and it can destroy the semantic meaning of the text. In other words, it only records how many times each word appears in the text and it does not give any importance to their order. Each word count becomes a dimension for that specific word.

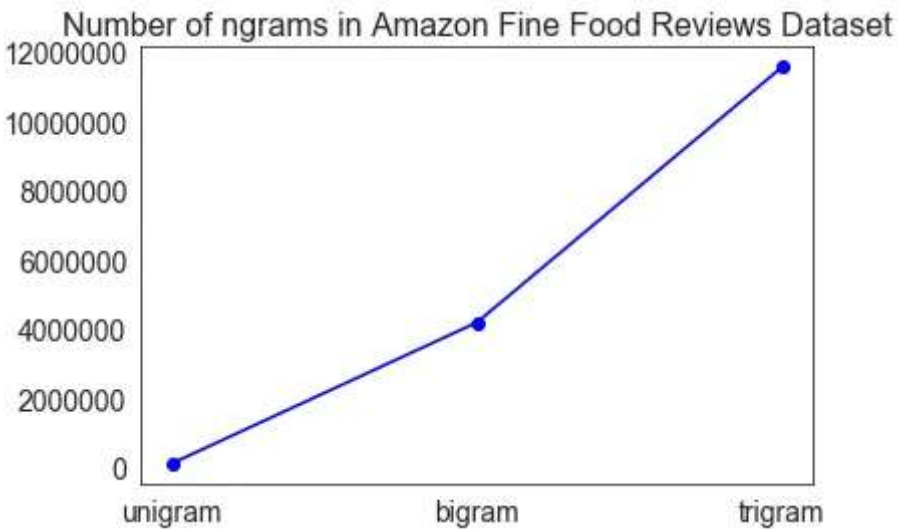
Bag of n-Grams

It is an extension of Bag-of-Words and represents n-grams as a sequence of n tokens. In other words, a word is 1-gram (unigram), two words are 2-grams (bigram), etc. It is applied in the NLP pipeline because it retains the original sequence of the text more than the Bag of Words representation. However, it has a very **high computational cost,** because theoretically k unique words can mean k^2 unique bigrams.

```
>>> bigram_converter = CountVectorizer(tokenizer=lambda doc: doc,
ngram_range=[2,2])

>>> trigram_converter = CountVectorizer(tokenizer=lambda doc: doc,
ngram_range=[3,3])
```

As can be seen below, in Amazon’s fine food data set the vocabulary length is of bag-of-words is 110,422 and becomes 4,189,296 for bag-of-biGrams and 11,572,565 for bag-of-triGrams.



Tf-Idf

Tf-Idf stands for term frequency-inverse document frequency, and instead of calculating the counts of each word in each document of the dataset (Bow), it

calculates the normalized count where each word count is divided by the number of documents this word appears.

Tf-idf(w, d)= Bow(w, d) * log(Total Number of Documents /(Number of documents in which word w appears))

> *If a word appears often in a particular document, but not in so many other documents, it is most likely that the word represents a particular meaning for that document and receives a larger count than before thanks to high Idf. On the other side, if a word appears in many documents, then its Idf is close to 1 and the logarithm turns 1 into 0 and decreases its effect.*

```
# Create the tf-idf representation using the bag-of-words matrix

>>> tfidf_transform = text.TfidfTransformer(norm=None)
>>> X_tfidf = tfidf_transform.fit_transform(X_bow)
```

Logistic Regression

After created a 70/30 train-test split of the dataset, I've applied logistic regression which is a classification algorithm used to solve binary classification problems. The logistic regression classifier uses the weighted combination of the input features and passes them through a sigmoid function. Sigmoid function transforms any real number input, to a number between 0 and 1.

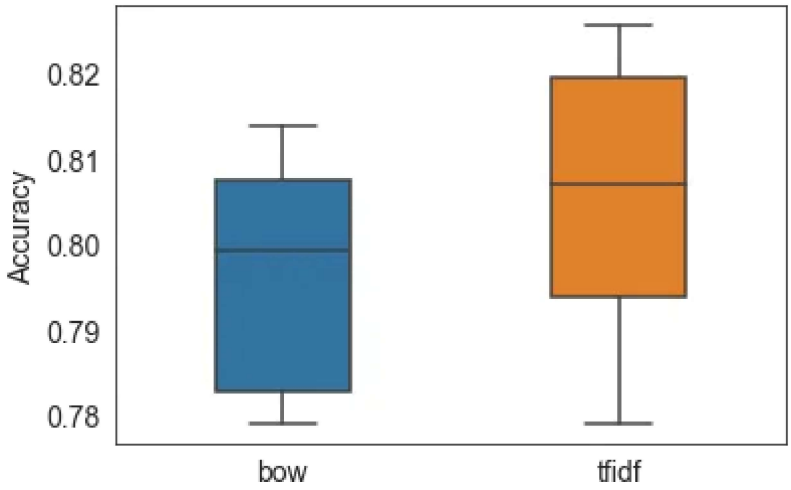
I have applied logistic regression classifier, on both Bag-of-triGrams and Tf-Idf features to compare their accuracy scores. Building the models on the default parameters gives us the accuracy scores as below:

```
Test Score with bow features 0.8083367181153533
Test Score with tf-idf features 0.8164601949634444
```

However, when the number of features is higher than the number of data points, the model tends to be underdetermined. To fix this problem, we need to introduce additional constraints that are known as **hyperparameters**. By using GridSearch I was able to try different combinations of values to find the model with the lowest

error metric, which is in this case log loss. In logistic regressions ‘C’ determines the amount of regularization, and the lower values increase regularization.

Building the logistic regression classifier with different regularization parameters using GridSearch, we’ve succeeded to improve accuracy scores.



Distribution of classifier accuracy for each feature set from 5-fold cross validation

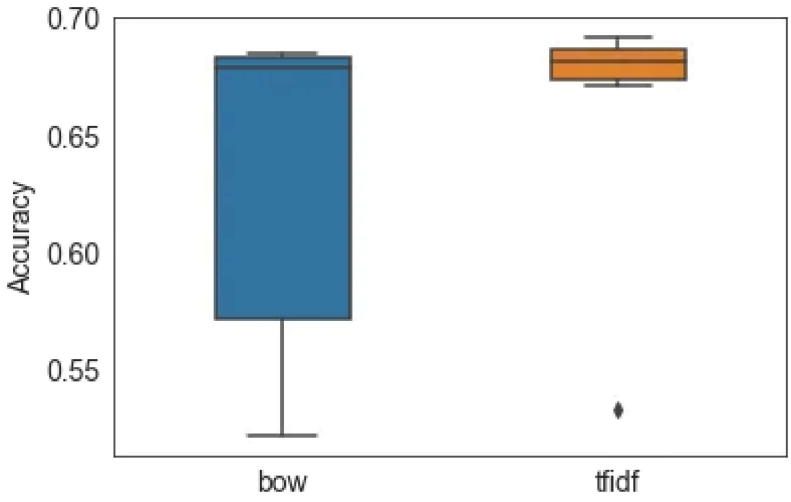
> *Note that, the low accuracy scores are due to bad regularization parameters. Once we have trained the model using the best hyperparameter we've reached the following accuracy scores.*

```
Test Score with bow features 0.8199465204440834
Test Score with tf-idf features 0.8244482805307338
```

As I have mentioned before, our machine learning model must have achieved a higher score than 78% in order to provide better performance than predicting the review always positive. By using tuned logistic regression classifier on both Bag-of-triGram and Tf-Idf, we have obtained successful results. It is also important to highlight that Tf-Idf featurization performed better compared to bag-of-trigrams which has also come with a very high computational cost.

Even though from this accuracy scores, it might have seemed that we did not increase the accuracy much... In order to evaluate this hypothesis, I have sampled the dataset to train the model by equally distributing the positive and negative reviews. Now we a **smaller dataset** but it has **balanced classes** so the starting probability is 50%!

I have applied logistic regression classifier with hyperparameters tuned by using GridSearch and I was able to obtain 71% accuracy which compared to 50% starting point is looking much better.



Distribution of classifier accuracy for each feature set from 5-fold cross validation

```
Test Score with bow features 0.7019666666666666
Test Score with tf-idf features 0.7100333333333333
```

Thank you for coming all the way here! I hope you enjoyed this article, and have fun practicing machine learning skills on text data!

If you liked this article, you can ***read my other articles here*** *and* ***follow me on Medium.*** Let me know if you have any questions or suggestions.✨

Enjoy this article? Become a member for more!

Text Classification NLP Feature Engineering Logistic Regression Customer Reviews

Follow

Written by Idil Ismiguzel

1.1K Followers · Writer for Analytics Vidhya

Data Scientist | Writing articles on Data Science & Machine Learning | MSc, MBA | https://de.linkedin.com/in/idilismiguzel

More from Idil Ismiguzel and Analytics Vidhya

Idil Ismiguzel in Towards Data Science

Imputing Missing Data with Simple and Advanced Techniques

A tutorial on mean, mode, time series, KNN, and MICE imputation

· 8 min read · May 12, 2022

372 4

Sajal Rastogi in Analytics Vidhya

Wifi -Hacking using PyWifi 🔐

.

4 min read · Feb 6, 2021

319 2

Yogesh Haribhau Kulkarni (PhD) in Analytics Vidhya

Microsoft AutoGen using Open Source Models

That means, without any Open AI API costs

4 min read · Oct 20

163 2

Idil Ismiguzel in Towards Data Science

A Guide to Association Rule Mining

Create insights from frequent patterns using market basket analysis with Python

· 10 min read · Apr 5

101 4

See all from Idil Ismiguzel

See all from Analytics Vidhya

Recommended from Medium

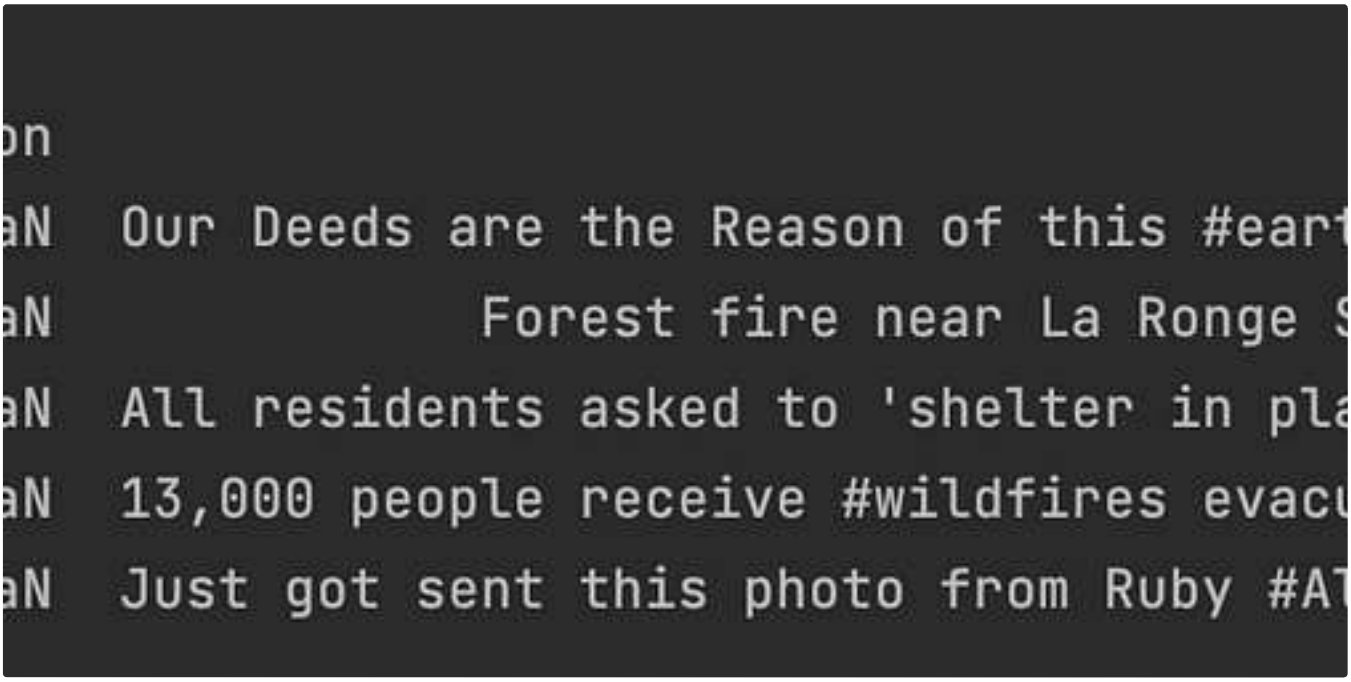
Khang Pham

Text Classification with BERT

In this tutorial, we will use BERT to develop your own text classification model.

8 min read · May 9

86 1



Hatice Şeyma Koç

Fasttext & Doc2Vec for Text Classification

Hello everyone,

5 min read · Jul 7

11

Lists

Natural Language Processing

803 stories · 369 saves

Practical Guides to Machine Learning

10 stories · 646 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 183 saves

Predictive Modeling w/ Python

20 stories · 565 saves

Saurav Agrawal

Multiclass Classification: OneVsRest and OneVsOne Classification Strategy

Disclaimer: Multiclass classification is supported with every classifier in scikit-learn out of the box. Only when experimentation with...

5 min read · Jun 23

4

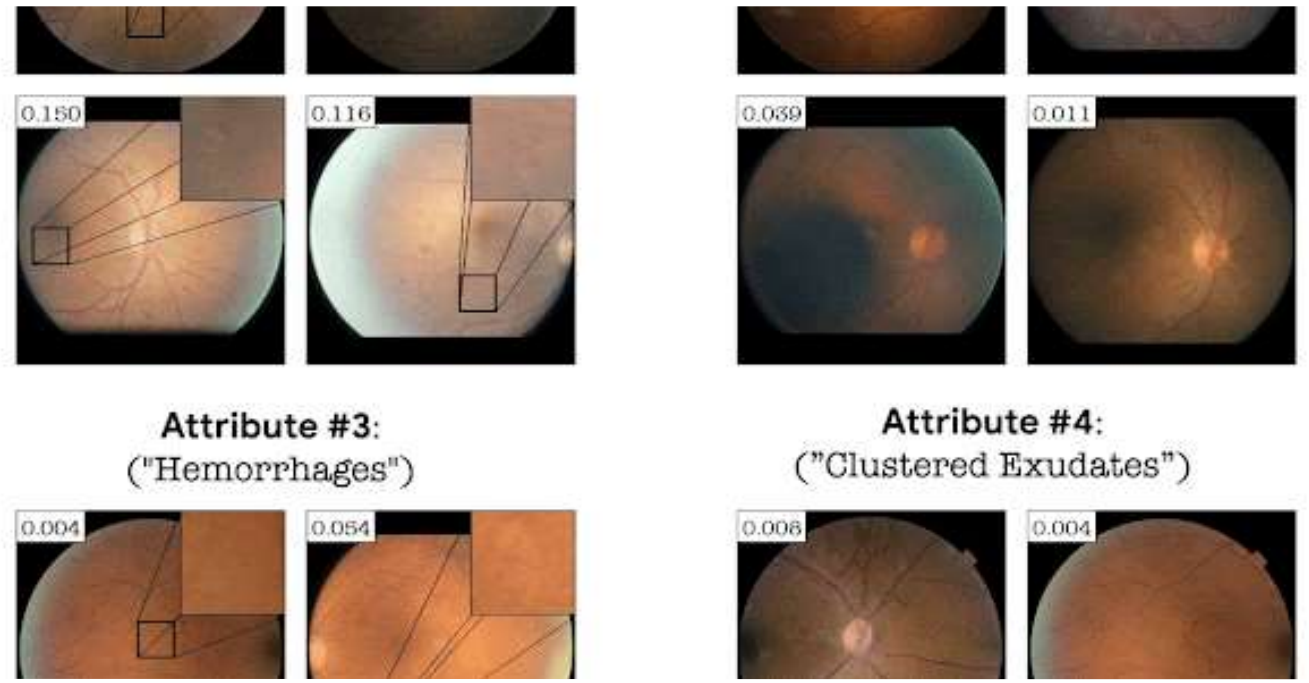
Mariya Mansurova in Towards Data Science

Topics per Class Using BERTopic

How to understand the differences in texts by categories

15 min read · Sep 9

404 1



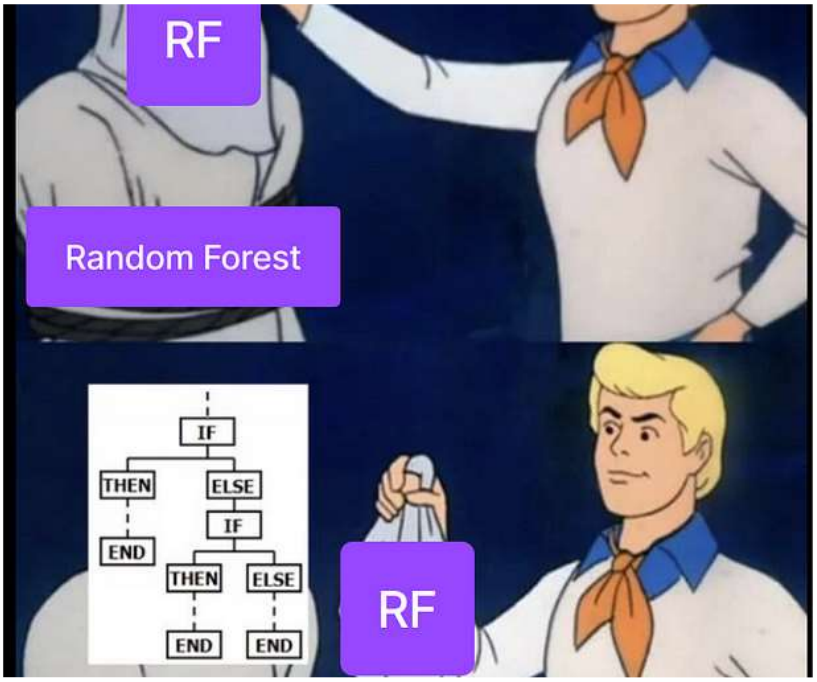
Everton Gomede, PhD

Multi-Label Classification in Python: Empowering Machine Learning with Versatility

Introduction

6 min read · May 25

23



Sunny Kumar

Random Forest Classification

Random forest is most popular machine learning algorithm which can be used in both classification and regression. It is easy and intuitive...

3 min read · May 21

52 1