Gradient Boosting

Theory and Intuition

fig ref: pierian data

Similar idea to AdaBoost, where weak learners are created in series in order to produce a strong ensemble model.

Uses residual error for learning.

fig ref: pierian data

Gradient Boosting vs. Adaboost:

- Larger Trees allowed in Gradient Boosting.
- Gradual series learning is based on training on the **residuals** of the previous model.

fig ref: pierian data

Area m^2	Bedrooms	Bathrooms	Price
200	3	2	$500,000
190	2	1	$462,000
230	3	3	$565,000

fig ref: pierian data

Train a decision tree on data

Area m^2	Bedrooms	Bathrooms	Price
200	3	2	$500,000
190	2	1	$462,000
230	3	3	$565,000

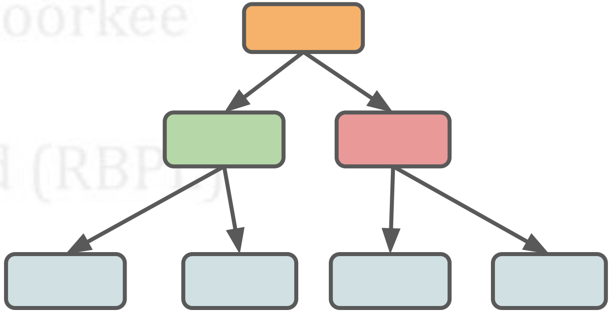


fig ref: pierian data

Get predicted ŷ value

y	ŷ
$500,000	$509,000
$462,000	$509,000
$565,000	$509,000

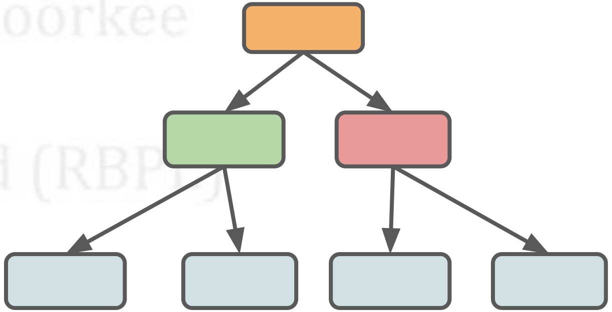


fig ref: pierian data

Residual: $\mathbf{e = y - \hat{y}}$

y	ŷ	e
\$500,000	\$509,000	-\$9,000
\$462,000	\$509,000	-\$47,000
\$565,000	\$509,000	\$56,000

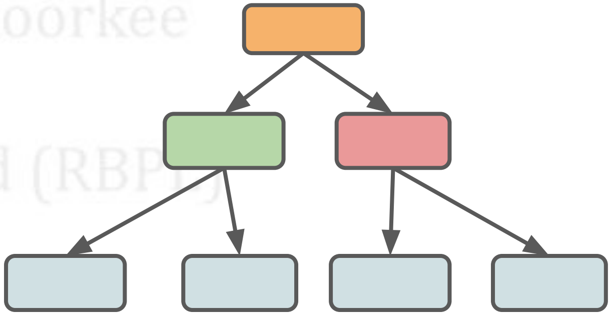


fig ref: pierian data

Create new model to predict the **error**

y	ŷ	e
$500,000	$509,000	-$9,000
$462,000	$509,000	-$47,000
$565,000	$509,000	$56,000

fig ref: pierian data

y	ŷ	e
$500,000	$509,000	-$9,000
$462,000	$509,000	-$47,000
$565,000	$509,000	$56,000

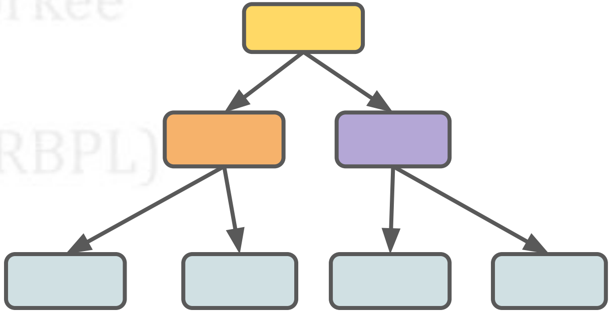


fig ref: pierian data

y	ŷ	e	f1
$500,000	$509,000	-$9,000	-$8,000
$462,000	$509,000	-$47,000	-$50,000
$565,000	$509,000	$56,000	$50,000

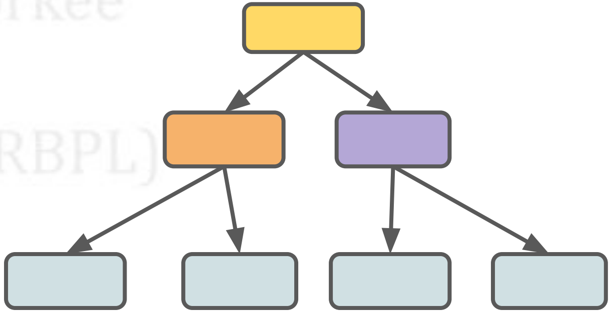


fig ref: pierian data

y	ŷ	e	f1
$500,000	$509,000	-$9,000	-$8,000
$462,000	$509,000	-$47,000	-$50,000
$565,000	$509,000	$56,000	$50,000

Area m^2	Bedrooms	Bathrooms
200	3	2
190	2	1
230	3	3

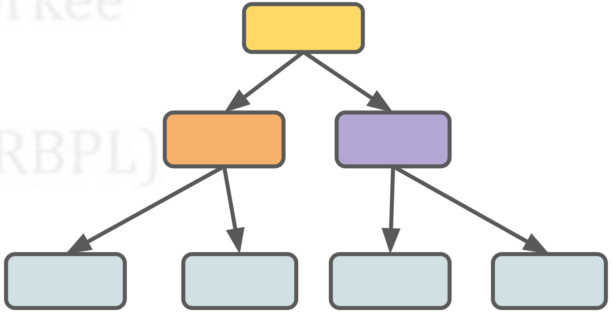


fig ref: pierian data

y	ŷ	e	f1
$500,000	$509,000	-$9,000	-$8,000
$462,000	$509,000	-$47,000	-$50,000
$565,000	$509,000	$56,000	$50,000

Area m^2	Bedrooms	Bathrooms
200	3	2
190	2	1
230	3	3

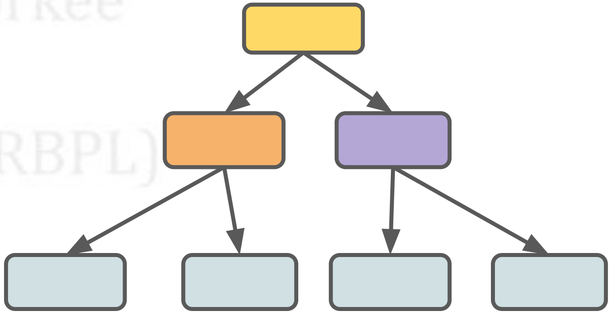


fig ref: pierian data

Update prediction using **error prediction**

y	ŷ	e	f1	F1 = ŷ + f1
$500,000	$509,000	-$9,000	-$8,000	
$462,000	$509,000	-$47,000	-$50,000	
$565,000	$509,000	$56,000	$50,000	

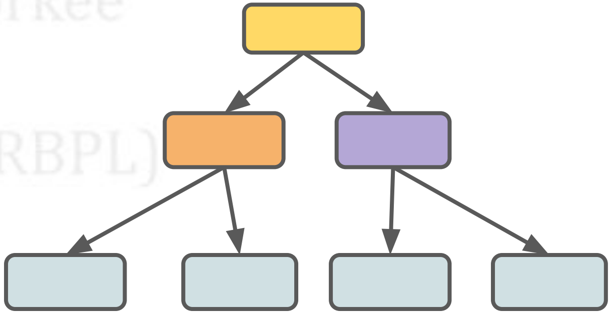
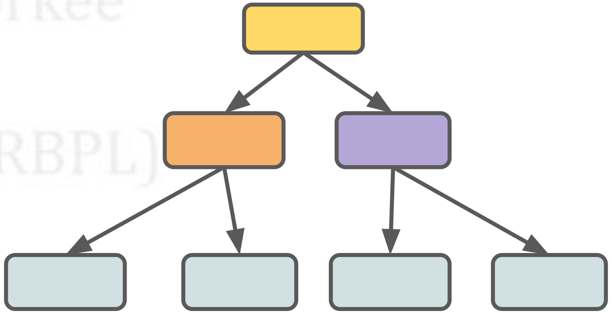


fig ref: pierian data

y	ŷ	e	f1	F1 = ŷ + f1
$500,000	$509,000	-$9,000	-$8,000	$501,000
$462,000	$509,000	-$47,000	-$50,000	$459,000
$565,000	$509,000	$56,000	$50,000	$559,000



Gradient Boosting Process

$$F_m = F_{m-1} + f_m$$

fig ref: pierian data

$$F_m = F_{m-1} + f_m$$

$$F_m = F_{m-1} + (\text{learning rate} * f_m)$$

fig ref: pierian data

Gradient Boosting Process

- Create initial model: f_0
- Train another model on error
  - $e = y - f_0$
- Create new prediction
  - $F_1 = f_0 + \eta f_1$
- Repeat as needed
  - $F_m = f_{m-1} + \eta f_m$

Note: for classification we can use the logit as an error metric:

$$\hat{y} = \log\left(\frac{\hat{p}}{1-\hat{p}}\right) \qquad \hat{p} = \frac{1}{1+e^{-\hat{y}}}$$

The learning rate is the same for each new model in the series and **not** unique to each subsequent model (unlike AdaBoost's alpha coefficient).

Gradient Boosting is fairly robust to overfitting, allowing for the number of estimators to be set high be default (~100).

fig ref: pierian data

Gradient Boosting Intuition

We optimize the series of trees by learning on the residuals, forcing subsequent trees to attempt to correct for the error in the previous trees.

fig ref: pierian data

The trade-off is training time.

A learning rate is between 0-1, which means a very low value would mean each subsequent tree has little “say”, meaning more trees need to be created, causing a longer computational training time.

Let’s explore Gradient Boosting in Jupyter Notebook!

An Introduction to Machine Learning with Python Programming
11 Sep 2023 - 20 Oct 2023

Naive Bayes and NLP

Conducted by:

and

Ritvij Bharat Private Limited (RBPL)

Presented by:
Shreyas Shukla

fig ref: pierian data

Using raw string text for machine learning models.

This idea in general is known as “Natural Language Processing”.

Overview

- Naive Bayes Algorithm and NLP
- Extracting Features from Text Data

Part One: Bayes’ Theorem

fig ref: pierian data

Naive Bayes is the shorthand for a set of algorithms that use Bayes' Theorem.

Bayes' Theorem leverages previously known probabilities to define probability of related events occuring.

fig ref: pierian data

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' Theorem.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

fig ref: pierian data

Bayes' Theorem

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

- **A** and **B** are events
- **P(A|B)** is probability of event **A** given that **B** is True.
- **P(B|A)** is probability of event **B** given that **A** is True.
- **P(A)** is probability of A occurring.
- **P(B)** is probability of B occurring.

fig ref: pierian data

Assume following situation:

- Every apartment in a building is fit with a fire alarm detection system.
- However, there are false alarms where smoke is detected but there is not a dangerous fire to put out (e.g. smoke from an oven).

fig ref: pierian data

The associated probabilities:

- Actual dangerous fires occur only 1% of the time.
- Smoke alarms are not good, and go off about 10% of the time.
- When there is an actual dangerous fire, 95% of the time the smoke alarms go off.

If you get a smoke alarm detecting a fire, what is the probability that there actually is a dangerous fire?

fig ref: pierian data

Event A: Dangerous Fire

Event B: Smoke Alarm Triggered

- P(A|B):
  - Probability of Fire given Smoke Alarm
- P(B|A):
  - Probability of Smoke Alarm given a dangerous fire

fig ref: pierian data

- Actual dangerous fires occur only 1% of the time. **P(Fire) = 1/100**
- Smoke alarms are not good and go off about 10% of the time. **P(Smoke) = 1/10**
- When there is an actual dangerous fire, 95% of the time the smoke alarms go off.
  - **P(Smoke|Fire) = 95/100**

fig ref: pierian data

Using Bayes' Theorem:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

P(Fire|Smoke) = P(Smoke|Fire)*P(Fire)/P(Smoke)

Using Bayes' Theorem:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

P(Fire|Smoke) = P(Smoke|Fire)*P(Fire)/P(Smoke)
P(Fire|Smoke) = 0.95 * 0.01 / 0.1
P(Fire|Smoke) = 0.095
P(Fire|Smoke) = 9.5%

fig ref: pierian data

Let’s see how Bayes’ Theorem can be extended to perform classification.

We’ll focus on using Bayes’ Theorem for Natural Language Processing Classification.

fig ref: pierian data

An Introduction to Machine Learning with Python Programming
11 Sep 2023 - 20 Oct 2023

Conducted by:
iHUB Divya Sampark, IIT Roorkee
and
Ritvij Bharat Private Limited (RBPL)

Part Two: Naive Bayes

Presented by:
Shreyas Shukla

fig ref: pierian data

Model the probability of belonging to a class given a vector of features.

$$\mathbf{x} = (x_1, \dots, x_n)$$

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \Rightarrow p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

fig ref: pierian data

The numerator is equivalent to a joint probability model:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})} \quad \Rightarrow \quad p(C_k \mid \mathbf{x}) = \frac{p(C_k, x_1, \dots, x_n)}{p(\mathbf{x})}$$

The chain rule can rewrite this numerator as a series of products of conditional probabilities:

$$
\begin{aligned}
p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2, \dots, x_n, C_k) \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k)\, p(x_3, \dots, x_n, C_k) \\
&= \cdots \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k) \cdots p(x_{n-1} \mid x_n, C_k)\, p(x_n \mid C_k)\, p(C_k)
\end{aligned}
$$

fig ref: pierian data

Finally we need to make an assumption that all x features are **mutually independent** of each other.

Thus, allowing for this conditional probability:

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$$

fig ref: pierian data

Then the joint model (the full Naive Bayes model) is fully written as:
(Where $\propto$ denotes proportionality)

$$\begin{aligned} p(C_k \mid x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &\propto p(C_k)\ p(x_1 \mid C_k)\ p(x_2 \mid C_k)\ p(x_3 \mid C_k)\ \cdots \\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,, \end{aligned}$$

Let’s walk through an example of using this Naive Bayes model.

Variations of Naive Bayes models, including:

- Multinomial Naive Bayes
- Gaussian Naive Bayes
- Complement Naive Bayes
- Bernoulli Naive Bayes
- Categorical Naive Bayes

Check documentation of sklearn!

fig ref: pierian data

We will focus on Multinomial Naive Bayes, since its used most often in the context of NLP.

Imagine we want to create a movie review aggregation website where we need to classify movie reviews into two categories: positive or negative.

fig ref: pierian data

Using previous reviews, we can have someone manually label them in order to create a labeled data set.

Then in the future, we could use our machine learning algorithm to automatically classify a new text review for us.

But how do we actually train on this text data?

Multinomial Bayes can work quite well with a simple count vectorization model (counting the frequency of each word in each document).

fig ref: pierian data

Start by separating out document classes:

fig ref: pierian data

Create “prior” probabilities for each class:

P(pos) = 25/35

P(neg) = 10/35

fig ref: pierian data

We will use these later!

P(pos) = 25/35

P(neg) = 10/35

Start with count vectorization on classes:

10	2	8	4
movie	actor	great	film

8	10	0	2
movie	actor	great	film

fig ref: pierian data

Calculate conditional probabilities:

10	2	8	4
movie	actor	great	film

P(movie|pos) = 10/24=0.42

8	10	0	2
movie	actor	great	film

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

8	10	0	2
movie	actor	great	film

$$P(\text{movie}|\text{pos}) = 10/24 = 0.42$$
$$P(\text{actor}|\text{pos}) = 2/24 = 0.08$$
$$P(\text{great}|\text{pos}) = 8/24 = 0.33$$
$$P(\text{film}|\text{pos}) = 4/24 = 0.17$$

fig ref: pierian data

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

P(movie|neg)= 8/20 = 0.4
P(actor|neg) = 10/20 = 0.5
P(great|neg) = 0/20 = 0
P(film|neg) = 2/20 = 0.1

fig ref: pierian data

Now a new review was created:

"movie actor"

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

fig ref: pierian data

Start with prior probability

25

0.42	0.08	0.33	0.17
10	2	8	4
movie	actor	great	film

10

0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

“movie actor”

P(pos) = (25/35)

fig ref: pierian data

Continue with conditional probabilities

0.42	0.08	0.33	0.17
10	2	8	4
movie	actor	great	film

“movie actor”

$$P(pos) \times P(movie|pos)$$

0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

fig ref: pierian data

25

0.42	0.08	0.33	0.17
10	2	8	4
movie	actor	great	film

10

0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

“movie actor”

$$P(pos) \times P(movie|pos) \times P(actor|pos)$$

25

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

"movie actor"

$$(0.71)\times(0.42)\times(0.08) = 0.024$$

10

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

fig ref: pierian data

Score is proportional to P(pos|“movie actor”)

25

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

“movie actor”

0.024 ∝ P(pos| “movie actor”)

10

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

fig ref: pierian data

Repeat same process with negative class

25

0.42	0.08	0.33	0.17
10	2	8	4
movie	actor	great	film

"movie actor"

P(neg)×P(movie|neg)×P(actor|neg)

10

0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

fig ref: pierian data

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

"movie actor"

$$(10/35) \times (0.4) \times (0.5) = 0.057$$

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

fig ref: pierian data

Score is proportional to P(neg|“movie actor”)

25

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

“movie actor”

0.057 ∝ P(neg| “movie actor”)

10

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

fig ref: pierian data

1. Compare both scores against each other
2. Classify based on highest score
3. Hence, this is classified as a negative review

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

“movie actor”

0.057 ∝ P(neg| “movie actor”)
0.024 ∝ P(pos| “movie actor”)

fig ref: pierian data

What about 0 count words?

25

0.42	0.08	0.33	0.17
10	2	8	4
movie	actor	great	film

"great movie"

10

0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

fig ref: pierian data

Probability is zero! Regardless of text!

0.42	**0.08**	**0.33**	**0.17**
10	2	8	4
movie	actor	great	film

0.4	**0.5**	**0**	**0.1**
8	10	0	2
movie	actor	great	film

"great movie"

$$P(\text{neg}) \times \boxed{P(\text{great}|\text{neg})} \times P(\text{movie}|\text{neg})$$

fig ref: pierian data

Alpha smoothing parameter

25

10**+1**	2**+1**	8**+1**	4**+1**
movie	actor	great	film

“great movie”

P(neg)×P(great|neg)×P(movie|neg)

10

8**+1**	10**+1**	0**+1**	2**+1**
movie	actor	great	film

fig ref: pierian data

Note how a higher alpha value will be more "smoothing", giving each word less distinct importance.

Now let's move on to focusing on feature extraction in general.

Are there better ways than just simply word frequency counts to extract features from text?

fig ref: pierian data

Extracting Features From Text Data

Theory and Intuition

fig ref: pierian data

Most classic ML algorithms can’t take in raw text as data.

Instead we need to perform a feature “extraction” from the raw text in order to pass numerical features to the ML algorithm.

fig ref: pierian data

Main Methods for Feature Extraction:

- Count Vectorization
- TF-IDF:
  - Term Frequency - Inverse Document Frequency

fig ref: pierian data

Count Vectorization

Create a vocabulary of all possible words

You are good
I feel good
I am good

fig ref: pierian data

Create a vocabulary of all possible words

YOU	ARE	GOOD	I	FEEL	AM

fig ref: pierian data

Create a vector of frequency counts

	YOU	ARE	GOOD	I	FEEL	AM
You are good	1	1	1	0	0	0
I feel good	0	0	1	1	1	0
I am good	0	0	1	1	0	1

fig ref: pierian data

```
messages = ["Hey, lets go to the game today!",
            "Call your sister.",
            "Want to go walk your dogs?"]
```


fig ref: pierian data

Document Term Matrix (DTM)

call	dogs	game	go	hey	lets	sister	the	to	today	walk	want	your
0	0	1	1	1	1	0	1	1	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	0	1	0	1	1	1

fig ref: pierian data

Count Vectorization treats every word as a feature, Frequency counts act as a “strength” of the feature/word.

For larger documents, matrices are stored as a **sparse matrix** to save space, since so many values will be zero.

fig ref: pierian data

Issues:

Very common words (e.g. “a” , “the”, “about”).

Words common to a particular set of documents (e.g. “run” in a set of different sports articles).

fig ref: pierian data

Stop Words are words common enough throughout a language that its usually safe to remove them.

Many NLP libraries have a built-in list of common stop words.

fig ref: pierian data

We can address the issue of document frequency by using a TF-IDF Vectorization process.

Instead of filling the DTM with word frequency counts it calculates term frequency-inverse document frequency value for each word(TF-IDF).

fig ref: pierian data

Term frequency **tf(t,d)**: is the raw count of a term in a document:

The number of times that term **t** occurs in document **d**.

fig ref: pierian data

Term Frequency alone isn’t enough for a thorough feature analysis of the text!

Let’s imagine very common terms, like “a” or “the”...

fig ref: pierian data

Because the term "the" is so common, term frequency will tend to incorrectly emphasize documents which happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms "red" and "dogs".

We also need to consider a group of documents where non stop words are common throughout all the documents

The word “run” in documents about various sports.

fig ref: pierian data

An inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient)

fig ref: pierian data

- The IDF is how common or rare a word is in the entire document set.
- The closer it is to 0, the more common a word is.
- Calculated by taking the **total number of documents**, **dividing it by the number of documents that contain a word**, and **calculating the logarithm.**

fig ref: pierian data

TF-IDF = term frequency * (1 / document frequency)

TF-IDF = term frequency * inverse document freq

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

fig ref: pierian data

Scikit-learn can calculate all these terms for us through the use of its API.

fig ref: pierian data

```python
from sklearn.feature_extraction.text import TfidfVectorizer

vect = TfidfVectorizer()
dtm = vect.fit_transform(messages)
```

call	dogs	game	go	hey	lets	sister	the	to	today	walk	want	your
0.000	0.00	0.403	0.307	0.403	0.403	0.000	0.403	0.307	0.403	0.00	0.00	0.000
0.623	0.00	0.000	0.000	0.000	0.000	0.623	0.000	0.000	0.000	0.00	0.00	0.474
0.000	0.46	0.000	0.349	0.000	0.000	0.000	0.000	0.349	0.000	0.46	0.46	0.349

fig ref: pierian data

Let’s Code !!

fig ref: pierian data

Extracting Features From Text Data

Understanding Core Concepts

Natural Language Processing

- Let’s begin understanding core concepts by manually creating a “bag of words” model.
- Recall that this is a frequency count of words in the documents.
- Let’s get started!

fig ref: pierian data

Extracting Features From Text Data

Utilizing Scikit-Learn

fig ref: pierian data

Classification with Text Data

Part One: Data Analysis and Features

fig ref: pierian data

Classification with Text Data

Part Two: Building Models

fig ref: pierian data

Text Classification Project Exercise

Solutions

fig ref: pierian data