MACHINE LEARNING

# All You Need to Know about Gradient Boosting Algorith 2. Classification

Algorithm explained with an example, math, and cod

Tomonori Masui

Feb 7, 2022 14 min read



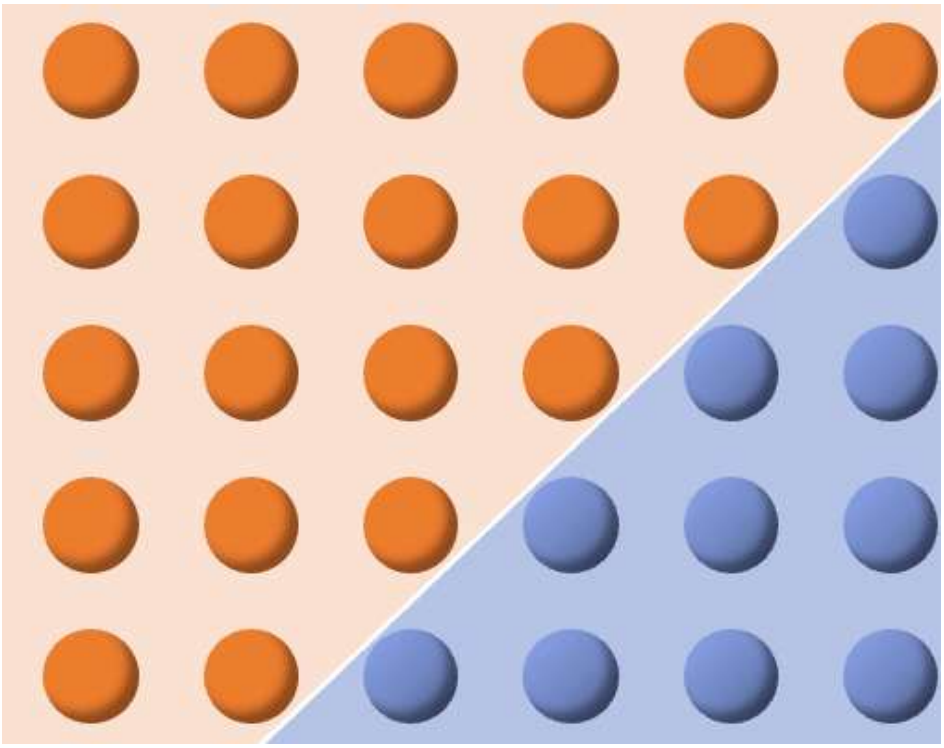


Image by author

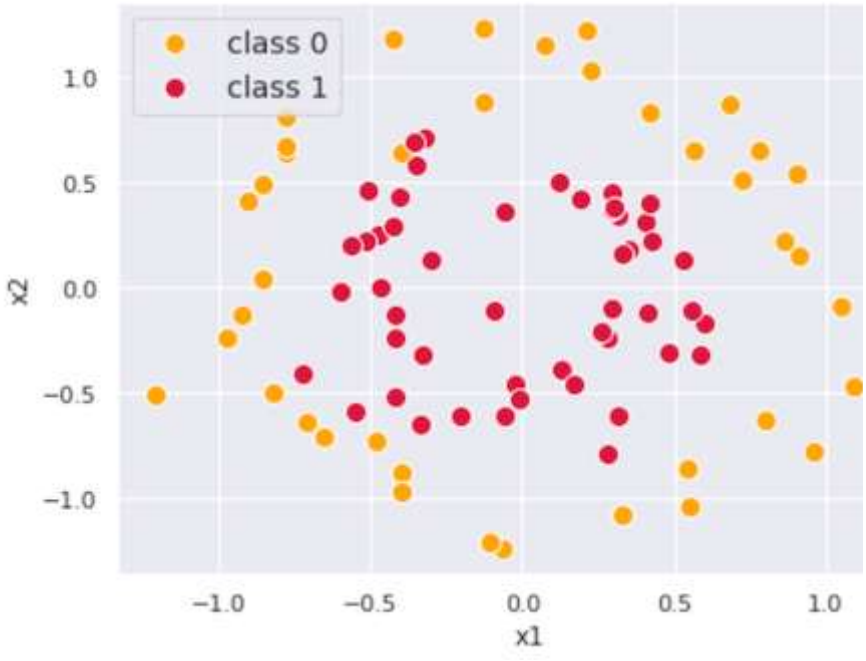
In the Part 1 article, we learned the gradient boo algorithm in its detail. As we reviewed in that pc is flexible enough to deal with any loss function

differentiable. That means if we just replace the loss function used for regression, specifically mean squared loss, with a loss function that deals with classification problems, we can perform classification without changing the algorithm itself. Even though the base algorithm is the same, there are still some differences that we want to know. In this post, we will dive into all the details of the classification algorithm.

## Algorithm with an Example

Gradient boosting is one of the variants of ensel
where you create multiple weak models (they ar
trees) and combine them to get better performa
In this section, we are building a gradient boosti
model using very simple example data to intuitiv
how it works.

The picture below shows the sample data. It ha
$y$ (0 and 1) and two features $x_1$ and $x_2$.

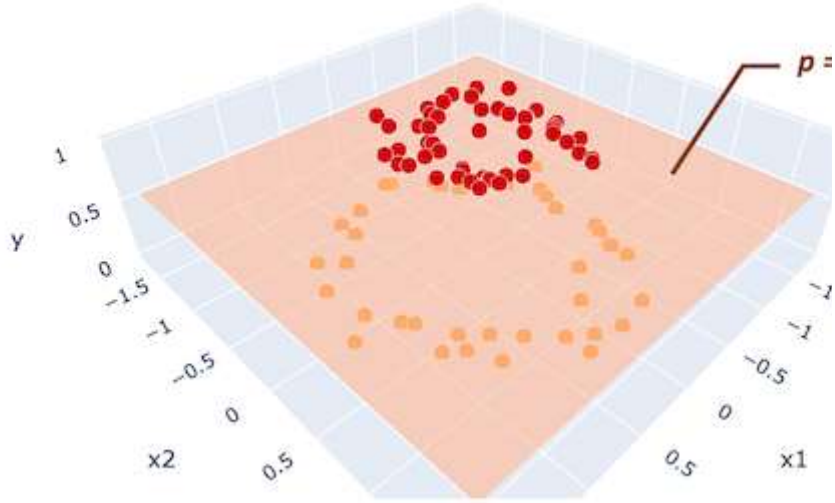


Sample for the classification problem (Image by author)

Our goal is to build a gradient boosting model that classifies those two classes. The first step is making a uniform prediction on a probability of class 1 (we will call it $p$) for all the data points. The most reasonable value for the uniform prediction might be the proportion of class 1 which is just a mean of $y$.

$$p = P(y = 1) = \bar{y}$$

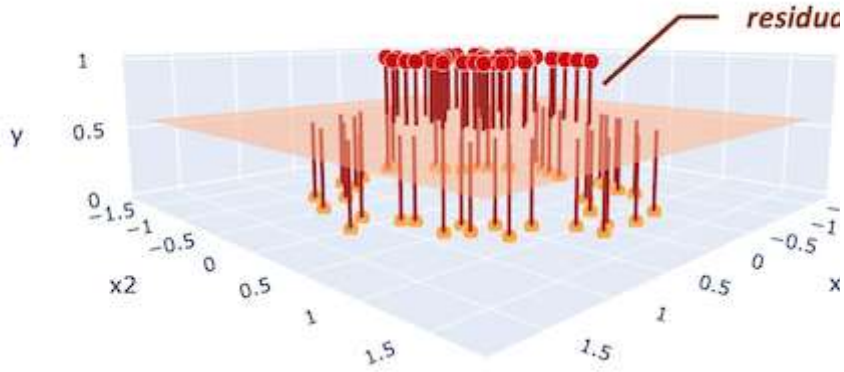
Here is a 3D representation of the data and the
At this moment, the prediction is just a plane th
uniform value $p = mean(y)$ on the $y$ axis all the t




Prediction plane (Image by author)

In our data, the mean of $y$ is 0.56. As it is bigge
everything is classified into class 1 with this init
Some of you might feel that this uniform value p
not make sense, but don't worry. We will improv
as we add more weak models to it.

To improve our prediction quality, we might want to focus on the residuals (i.e. prediction error) from our initial prediction as that is what we want to minimize. The residuals are defined as $r_i = y_i - p$ ($i$ represents the index of each data point). In the figure below, the residuals are shown as the brown lines that are the perpendicular lines from each data point to the prediction plane.



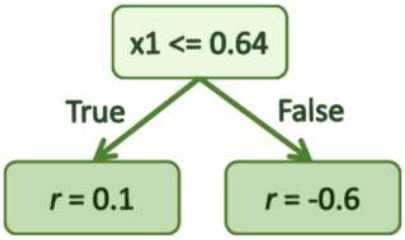

Residuals (Image by author)

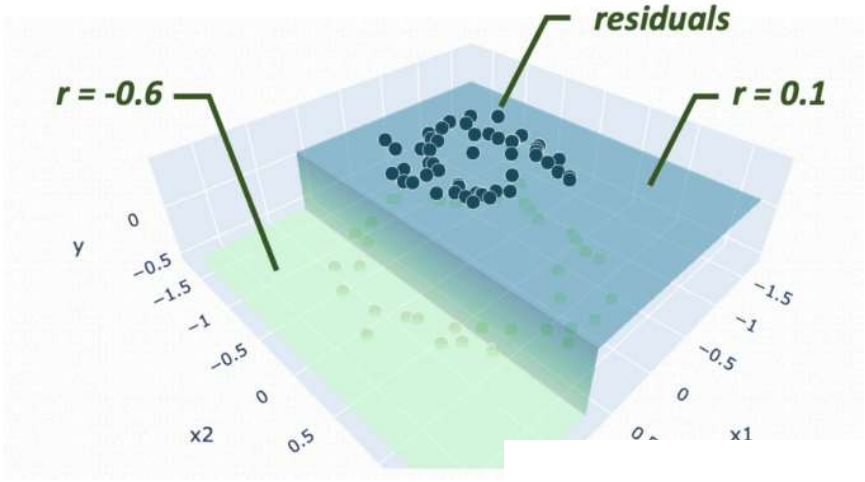
LATEST
EDITOR'S PICKS
DEEP DIVES
NEWSLETTER
WRITE FOR TDS
Sign in
Submit an Article

To minimize these residuals, we are building a r model with both $x_1$ and $x_2$ as its features and t its target. If we can build a tree that finds some between $x$ and $r$, we can reduce the residuals prediction $p$ by utilizing those found patterns.

To simplify the demonstration, we are building v each of that only has one split and two termina called "stump". Please note that gradient boosti have a little deeper trees such as ones with 8 tc nodes.

Here we are creating the first tree predicting the two different values $r = \{0.1, -0.6\}$.




Created tree (Image by author)



You might now think we want to add these pred
our initial prediction $p$ to reduce its residuals if
the post talking about the regression algorithm,
slightly different with classification. The values
gamma) that we are adding to our initial predict
in the following formula:

$$\gamma_j = \frac{\sum_{x_i \in R_j} (y_i - p)}{\sum_{x_i \in R_j} p(1-p)}$$

$\gamma_j$ is computed for each terminal node $j$

Aggregating for all the belongs to terminal nc

$\sum_{x_i \in R_j}$ *means we are aggregating the values in the sample* $x_i$ *s that belong to the terminal node the index of each terminal node. You might notic numerator of the fraction is the sum of the resid terminal node* $j$*. We will go through all the calc us this formula in the next section, but let's just* $\gamma$ *for now. Below is the computed values of* $\gamma_1$ *a*

$$\gamma_1 = \frac{\sum_{x_i \in R_1} (y_i - 0.56)}{\sum_{x_i \in R_1} 0.56 \cdot (1 - 0.56)} = 0.3$$

$$\gamma_2 = \frac{\sum_{x_i \in R_2} (y_i - 0.56)}{\sum_{x_i \in R_2} 0.56 \cdot (1 - 0.56)} = -2.2$$

This $\gamma$ is not simply added to our initial predicti are converting $p$ into log-odds (we will call this converted value $F(x)$), then adding $\gamma$ to it. For t familiar with log-odds, it is defined below. You r used in logistic regression.

$$log(odds) = log\left(\frac{p}{1-p}\right)$$

log-odds

One more tweak on the prediction update is tha down by **learning rate** $\nu$, which ranges between is added to the log-odds-converted prediction the model not to overfit the training data.

$$F_1(x) = F_0(x) + \nu \cdot \gamma$$

Updated prediction — Initial prediction — Learning rate

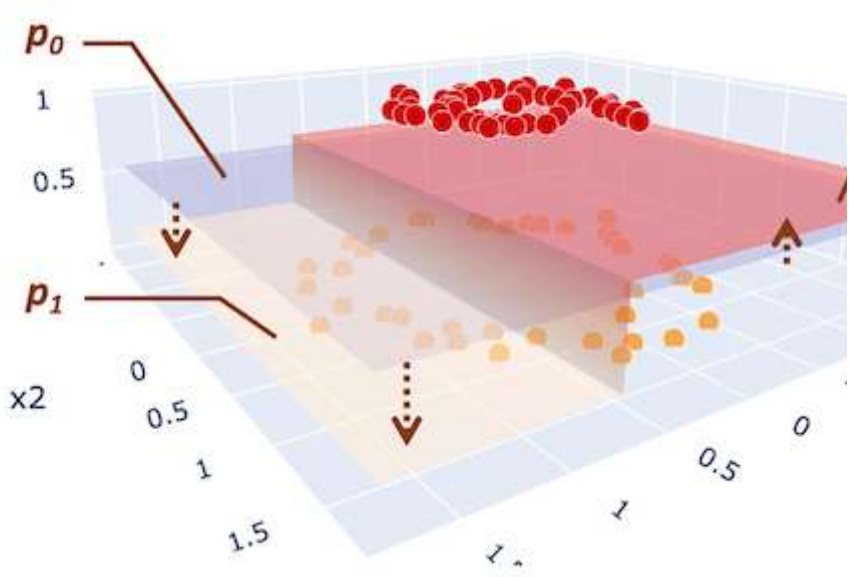
In this example, we use a relatively big learning make the optimization process easier to unders usually supposed to be much smaller values su

By substituting actual values for the variables in the right side of the above equation, we get our updated prediction $F_1(x)$.

$$F_1(x) = \begin{cases} log\left(\frac{0.56}{1-0.56}\right) + 0.9 \cdot 0.3 = 0.5 & if\ x_1 \le 0.64 \\ log\left(\frac{0.56}{1-0.56}\right) - 0.9 \cdot 2.2 = -1.7 & otherwise \end{cases}$$

If we convert log-odds $F(x)$ back into the predic
$p(x)$ (we will cover how we can convert it in the
looks like a stair-like object below.

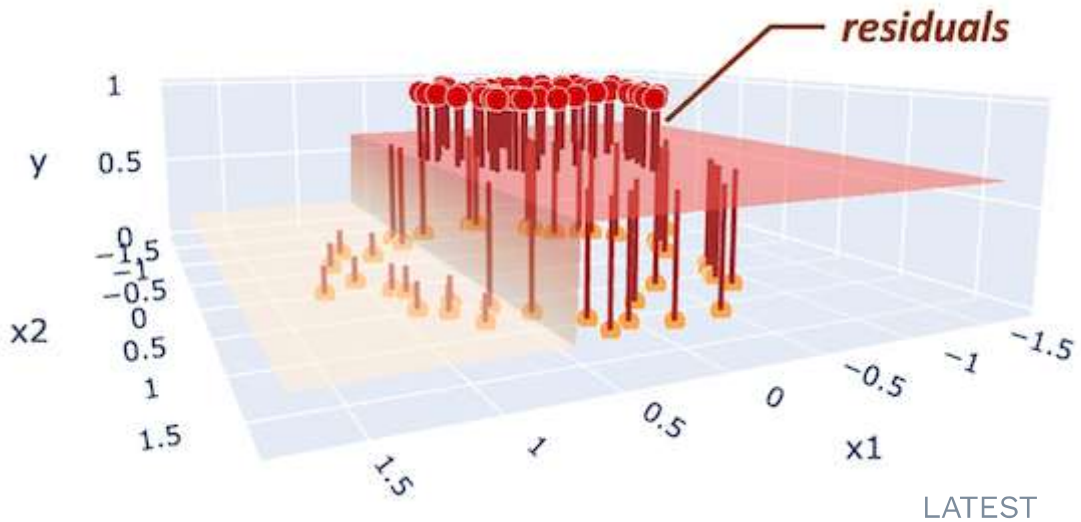





Updated prediction plane (Image by author)

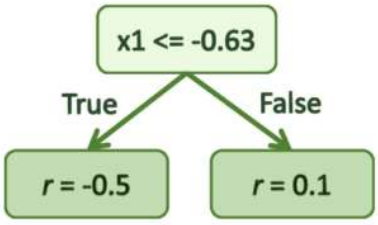
The purple-colored plane is the initial predictio
*updated to the red and yellow plane* $p_1$_.

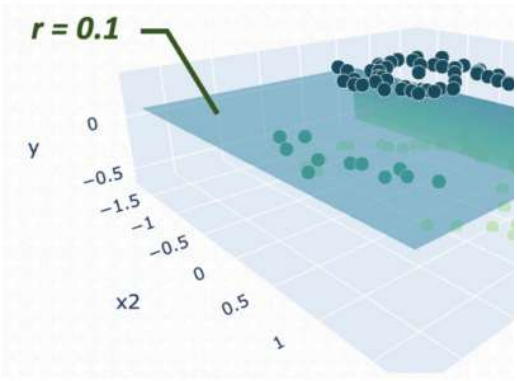
Now, the updated residuals $r$ looks like this:




Updated residuals (Image by author)

In the next step, we are creating a regression tr
same $x_1$ and $x_2$ as the features and the update
its target. Here is the created tree:




Created tree (Image by author)

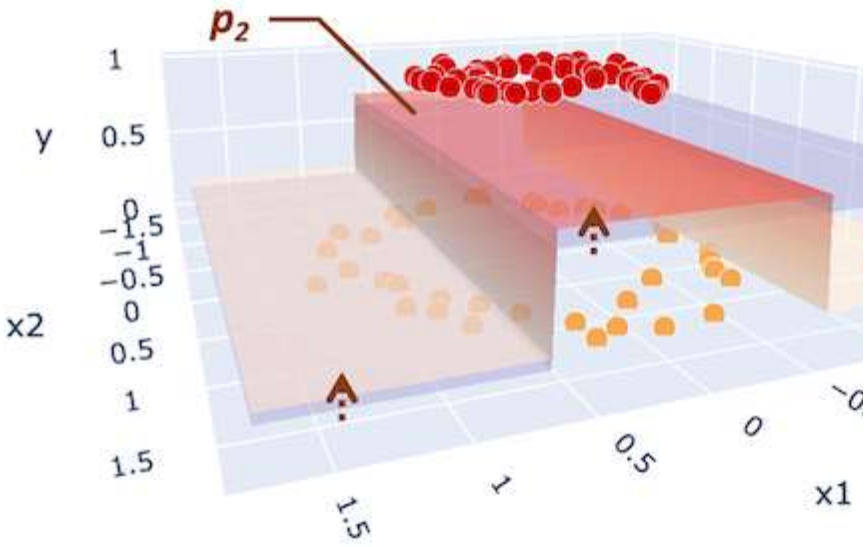
We apply the same formula to compute $\gamma$. The
along with the updated prediction $F_2(x)$ are as f

$$F_2(x) = \begin{cases} F_1(x) - \nu \cdot 2.3 = 0.5 - 0.9 \cdot 2.3 = -1.6 & if\ x_1 \le -0.63 \\ F_1(x) + \nu \cdot 0.4 = 0.5 + 0.9 \cdot 0.4 = 0.9 & else\ if\ -0.63 < x_1 \le 0.64 \\ F_1(x) + \nu \cdot 0.4 = -1.7 + 0.9 \cdot 0.4 = -1.3 & otherwise \end{cases}$$

These are $\gamma$ computed with this formula:

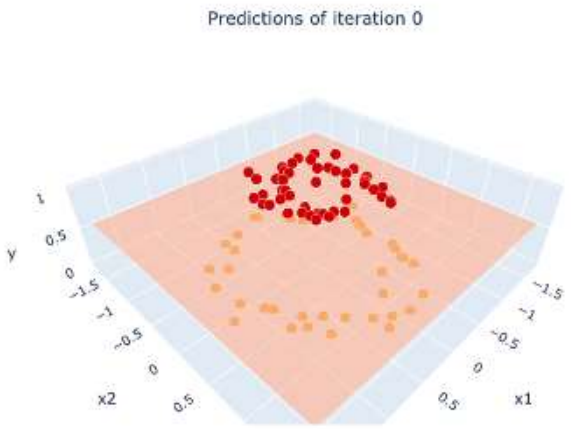
$$\gamma_j = \frac{\sum_{x_i \in R_j} (y_i - p)}{\sum_{x_i \in R_j} p(1-p)}$$

Again, if we convert log-odds $F_2(x)$ back into th probability $p_2(x)$, it looks like something below.



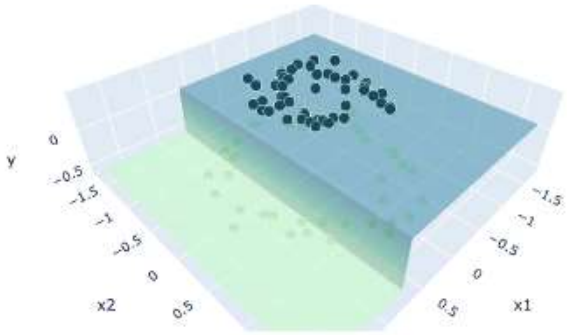

Updated prediction plane (Image by author)

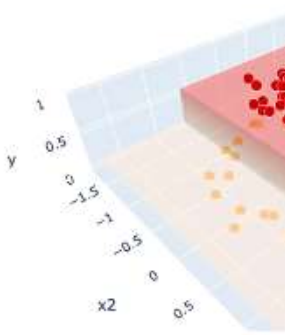
We iterate these steps until the model predictic improving. The figures below show the optimiza 0 to 4 iterations.




Residuals of iteration 1

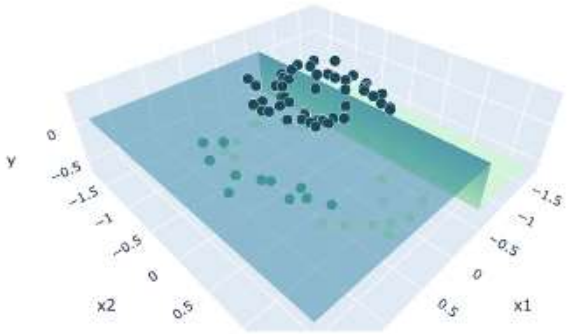
Predictio

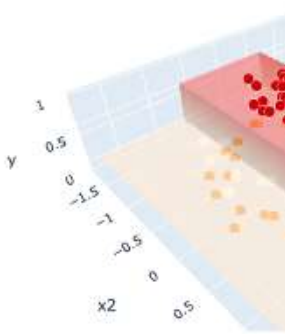



Residuals of iteration 2

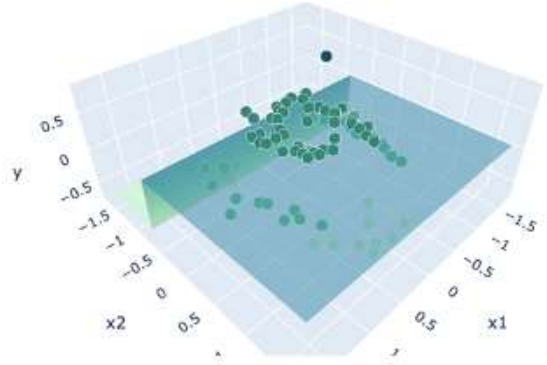
Predictio

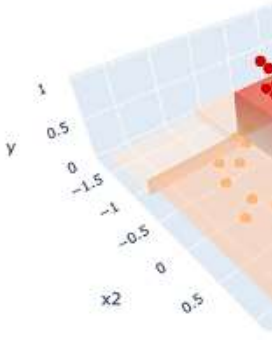



Residuals of iteration 3

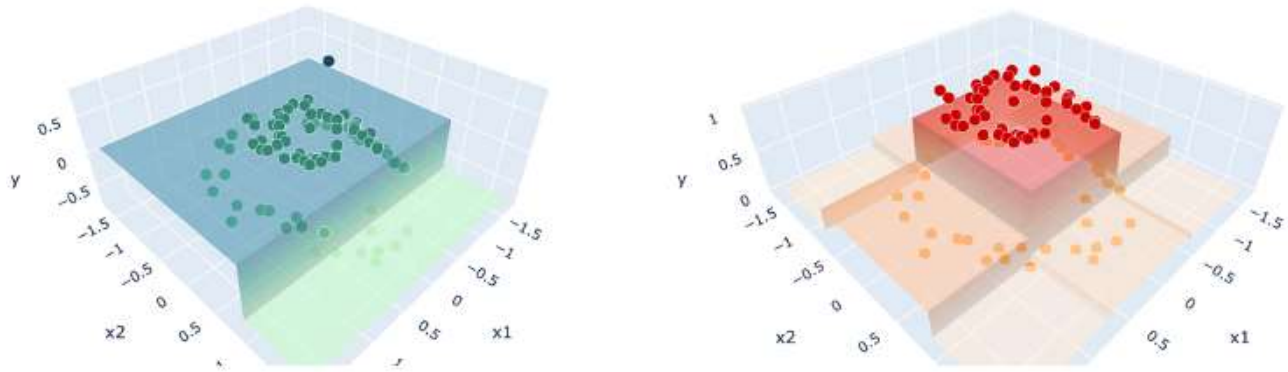
Predic




Residuals of iteration 4

Predi

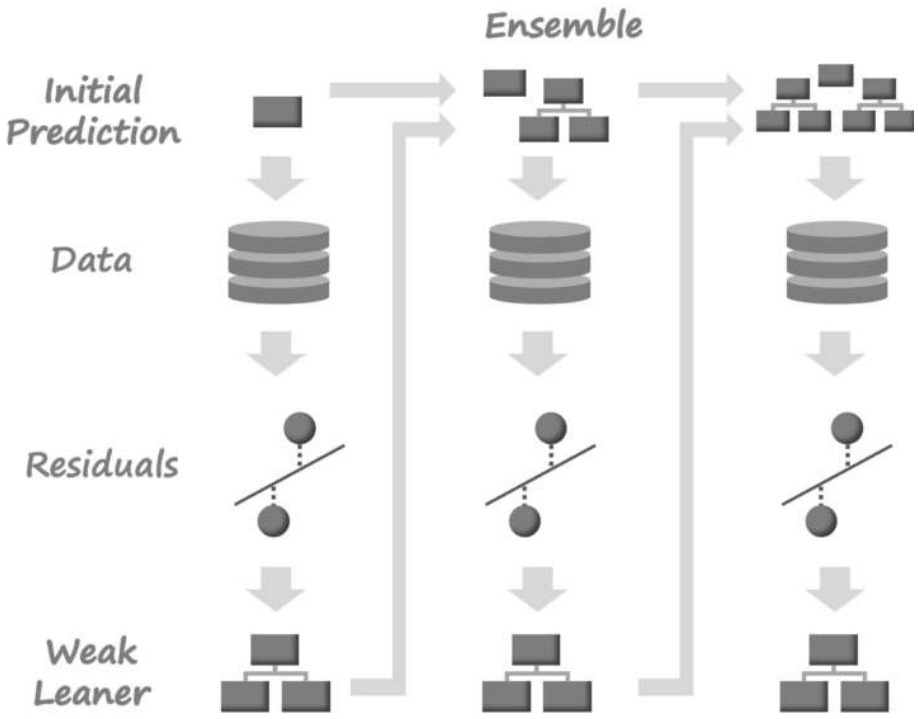



Prediction update through iterations (Image by auth

You can see the combined prediction $p(x)$ (red
is getting closer to our target $y$ as we add more
combined model. This is how gradient boosting
complex targets by combining multiple weak mo

The image below summarizes the whole proces






Process of the algorithm (Image by Author)

## Math

In this section, we are learning a more generalized algorithm by looking at its math details. Here is the whole algorithm in math formulas.

**Gradient Boosting Algorithm**

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. for $m = 1$ *to* $M$:

2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}}$

2-2. Train regression tree with features $x$ against $r$ and c

reasions $R_{jm}$ for $j = 1,\ldots, J_m$

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ $f$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

Source: adapted from Wikipedia and Friedman's pap

Let's take a close look at it line by line.

### Step 1

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

The first step is creating an initial constant prec

$L$ is the loss function and we are using log loss

generally called cross-entropy loss) for it.

$$L = -(y_i \cdot log(p) + (1 - y_i) \cdot log(1 - p))$$

Log Loss

$y_i$ is our classification target and it is either 0 or 1. $p$ is the predicted probability of class 1. You might see $L$ taking different values depending on the target class $y_i$.

$$L = \begin{cases} -log(p) & if\ y_i = 1 \\ -log(1-p) & if\ y_i = 0 \end{cases}$$

As $-log(x)$ is the decreasing function of $x$, the b prediction (i.e. increasing $p$ for $y_{i=1}$), the smalle have.

$argmin$ means we are searching for the value $\gamma$ ( minimizes $\_\Sigma L(y\_i\_,\gamma)$. *While it is more straightfo* $\gamma$ *is the predicted probability* $p$*, we assume* $\gamma$ *is* ***** it makes all the following computations ea who forgot the log-odds definition which we revi section, it is defined as* $\log(odds) = \log(p/(1-p))\_$.

To be able to solve the $argmin$ problem in terms are transforming the loss function into the func

$$L = -(y_i \cdot log(p) + (1 - y_i) \cdot log(1 - p))$$

$$= -(y_i \cdot (log(p) - log(1 - p)) + log(1 - p))$$

$$= -\left(y_i \cdot log\left(\frac{p}{1 - p}\right) + log(1 - p)\right)$$

$$= -(y_i \cdot log(odds) + log(1 - p))$$

Now, we might want to replace $p$ in the above e
something that is expressed in terms of log-odc
transforming the log-odds expression shown ea
represented by log-odds:

$$log\left(\frac{p}{1 - p}\right) = log(odds)$$

$$\frac{p}{1 - p} = e^{log(odds)}$$

$$p = (1 - p)e^{log(odds)}$$

$$\left(1 + e^{log(odds)}\right)p = e^{log(odds)}$$

$$p = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

Then, we are substituting this value for $p$ in the equation and simplying it.

p is replaced with *this*

$$L = -\left(y_i \cdot log(odds) + log\left(1 - \frac{e^{log(odds)}}{1 + e^{log(odds)}}\right)\right)$$

$$= -\left(y_i \cdot log(odds) + log\left(\frac{1}{1 + e^{log(odds)}}\right)\right)$$

$$= -\left(y_i \cdot log(odds) + \underbrace{log(1)}_{=0} - log\left(1 + e^{log(odds)}\right)\right)$$

$$= -\left(y_i \cdot log(odds) - log\left(1 + e^{log(odds)}\right)\right)$$



Now we are finding $\gamma$ (please remember we are log-odds) _ *that minimizes* $\Sigma L$_. We are taking a with respect to log-odds.

$$\frac{\partial}{\partial log(odds)} \sum_{i=1}^{n} L = -\frac{\partial}{\partial log(odds)} \sum_{i=1}^{n} \left[y_i \cdot log(odds) - \right.$$

$$= -\sum_{i=1}^{n} y_i + n \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

$$= -\sum_{i=1}^{n} y_i + np$$

In the equations above, we replaced the fractior odds with $p$ to simplify the equation. Next, we $\partial log(odds)$ equal to 0 and solving it for $p$.

$$-\sum_{i=1}^{n} y_i + np = 0$$

$$np = \sum_{i=1}^{n} y_i$$

$$p = \frac{1}{n}\sum_{i=1}^{n} y_i = \bar{y}$$

In this binary classification problem, $y$ is either
mean of $y$ is actually the proportion of class 1. `
see why we used $p = mean(y)$ for our initial predi

As $\gamma$ is log-odds instead of probability $p$, we ar
into log-odds.

$$F_0(x) = \overset{*}{\gamma} = log\left(\frac{\bar{y}}{1-\bar{y}}\right)$$



**Step2**

> 2. for $m = 1$ to $M$:

The whole step2 processes from 2–1 to 2–4 are
$M$ denotes the number of trees we are creating
represents the index of each tree.

**Step 2–1**

> 2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$

We are calculating residuals $r_{im}$ by taking a derivative of the loss function with respect to the previous prediction $F_{m-1}$ and multiplying it by –1. As you can see in the subscript index, $r_{im}$ is computed for each single sample i. Some of you might be wondering why we are calling this $r_{im}$ residuals. This value is actually negative gradient that gives us the directions (+/–) and the magnitude in which the loss function can be minimized. You will see why we are calling it residuals shortly. E technique where you use a gradient to minimize model is very similar to gradient descent techni typically used to optimize neural networks. (In f slightly different from each other. If you are inte look at this article detailing that topic.)

LATEST

EDITOR'S PICKS

DEEP DIVES

NEWSLETTER

WRITE FOR TDS

Sign in

Submit an Article

Let's compute the residuals here. $F_{m-1}$ in the e the prediction from the previous step. In this fir $F_0$. As in the previous step, we are taking a der respect to log-odds instead of p since our predi odds. Below we are using L " expressed by log-c in the previous step.

$$r_{im} = -\frac{\partial}{\partial log(odds)} L$$

$$= \frac{\partial}{\partial log(odds)} \left[ y_i \cdot log(odds) - log\left(1 + \right.\right.$$

$$= y_i - \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

In the previous step, we also got this equation:

$$p = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

So, we can replace the second term in $r_{im}$ equation with p .

$$r_{im} = y_i - p$$

You might now see why we call $r$ residuals. This interesting insight that the negative gradient tha direction and the magnitude to which the loss i actually just residuals.



**Step 2–2**

2-2. Train regression tree with features $x$ against $r$ and cr
reasions $R_{jm}$ for $j = 1,\ldots,J_m$

$j$ represents a terminal node (i.e. leave) in the t the tree index, and capital $J$ means the total nu

**Step 2–3**

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ fo

We are searching for $\gamma_{jm}$ that minimizes the los each terminal node j . $\Sigma_{x_i \in R_{jm}} L(m)$ *means we are loss on all the $x_i$s that belong to the terminal nc* plugin the loss function into the equation.

$$\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

$$= \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} -\left(y(F_{m-1}(x_i) + \gamma) - log\left(1 + e^{F_{m-1}(x_i)+\gamma}\right)\right)$$

Solving this equation for $\gamma_j m$ is going to be very hard. To make it more easily solvable, we are making the approximation of L " by using the second-order Taylor polynomial. A Tayl way to approximate any function as a polynomia infinite/finite number of terms. While we are no details here, you can look at this tutorial which nicely if you are interested.

Here is the approximation of $L$ using the second polynomial:

$$L(y, F_{m-1}(x_i) + \gamma) \approx L(y, F_{m-1}(x_i)) + \frac{\partial}{\partial F} L(y_i, F_{m-1}(x_i))\gamma +$$

We are substituting this approximation for $L$ in $\gamma_j m$, then finding the value of γ_ $_j m$ t _hat make Σ(*) equal zero.

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jm}} \left( \underbrace{L(y_i, F_{m-1}(x_i))}_{\text{Derivative of this term is zero}} + \frac{\partial}{\partial F} L(y_i, F_{m-1}(x_i))\gamma + \frac{1}{2} \frac{\partial^2 L(y_i, F_{m-1}(x_i))}{\partial F^2} \gamma^2 \right) = 0$$

$$\sum_{x_i \in R_{jm}} \left( \frac{\partial}{\partial F} L(y_i, F_{m-1}(x_i)) + \frac{\partial^2 L(y_i, F_{m-1}(x_i))}{\partial F^2} \gamma \right) = 0$$

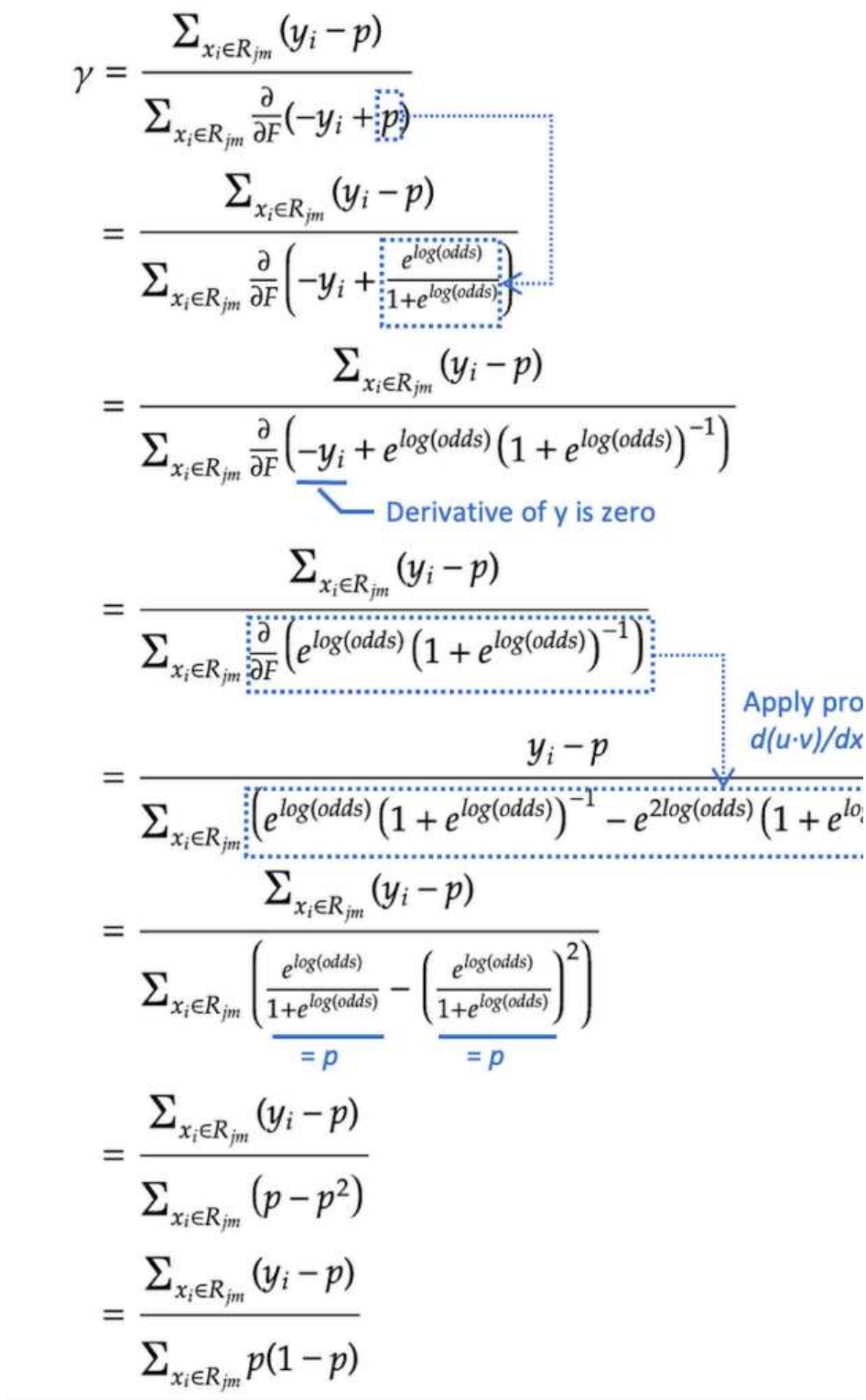
$$\sum_{x_i \in R_{jm}} \frac{\partial^2 L(y_i, F_{m-1}(x_i))}{\partial F^2} \gamma = - \sum_{x_i \in R_{jm}} \frac{\partial}{\partial F} L(y_i, F_{m-1}(x_i))$$

$$\gamma = \frac{-\sum_{x_i \in R_{jm}} \frac{\partial}{\partial F} L(y_i, F_{m-1}(x_i))}{\sum_{x_i \in R_{jm}} \frac{\partial^2}{\partial F^2} L(y_i, F_{m-1}(x_i))}$$

As we already computed $\partial L/\partial F$ in the previous s

$$\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)} = -(y_i - p)$$

We are substituting this for $\partial L/\partial F$ in the $\gamma$ equa




Finally, we got this simplified equation for the v
we used in the previous section.

**Step 2–4**

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

In the final step, we are updating the prediction of the combined model $F_m$. $\gamma_{jm} 1(x \in R_{jm})$ means that we pick the value $\gamma_{jm}$ if a given x falls in a terminal node $R_{jm}$. As all the terminal nodes are exclusive, any given single x falls into only a single terminal node and corresponding $\gamma_{jm}$ is added to the previous and then it makes the updated prediction $F_m$."

As mentioned in the previous section, $\nu$ is learr between 0 and 1 which controls the degree of c additional tree prediction $\gamma$ to the combined pr smaller learning rate reduces the effect of the a prediction, but it basically also reduces the cha overfitting to the training data.

---

Now we have gone through the whole steps. To model performance, we want to iterate step 2 $M$ means adding $M$ trees to the combined model. I might often want to add more than 100 trees to model performance.

If you read my article for a regression algorithm, the math computation of the classification algo more complicated than the regression. While ar derivative computation of the log-loss function the underlining math algorithm which is shown of this section is exactly the same. That is actua

of the gradient boosting algorithm as exactly the same algorithm works for any loss function as long as it is differentiable. In fact, popular gradient boosting implementations such as [XGBoost](https://xgboost.readthedocs.io/en/stable/parameter.html#learning-task-parameters) or [LightGBM](https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective) have a wide variety of loss functions, so you can choose whatever loss functions that suit your problem
loss functions available in XGBoost or LightGBM



## Code

In this section, we are translating the maths we
into a viable python code to help us understand
further. We are using `DecisionTreeRegressor` from s
build trees which helps us just focus on the gra
algorithm itself instead of the tree algorithm. W
scikit-learn style implementation where you tra
`fit` method and make predictions with `predict`

```
class CustomGradientBoostingClassifier:

    def __init__(self, learning_rate, n_estimators, max_depth=1):
        self.learning_rate = learning_rate
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.trees = []

    def fit(self, X, y):

        F0 = np.log(y.mean()/(1-y.mean()))  # log-odds values
        self.F0 = np.full(len(y), F0)  # converting to array with
        Fm = self.F0.copy()

        for _ in range(self.n_estimators):
```

```
            p = np.exp(Fm) / (1 + np.exp(Fm))  # converting back to probabilities
            r = y - p  # residuals
            tree = DecisionTreeRegressor(max_depth=self.max_depth, random_state=0)
            tree.fit(X, r)
            ids = tree.apply(x)  # getting the terminal node IDs

            # looping through the terminal nodes
            for j in np.unique(ids):
              fltr = ids == j

              # getting gamma using the formula (Σresiduals/Σp(1-
              num = r[fltr].sum()
              den = (p[fltr]*(1-p[fltr])).sum()
              gamma = num / den

              # updating the prediction
              Fm[fltr] += self.learning_rate * gamma

              # replacing the prediction value in the tree
              tree.tree_.value[j, 0, 0] = gamma

            self.trees.append(tree)

    def predict_proba(self, X):

        Fm = self.F0

        for i in range(self.n_estimators):
            Fm += self.learning_rate * self.trees[i].predict(X)

        return np.exp(Fm) / (1 + np.exp(Fm))  # converting back i
```

custom_gbdt.py hosted with ❤ by GitHub



Please note that all the trained trees are stored object and it is retrieved when we make predict `predict_proba` method.

Next, we are checking if our `CustomGradientBoostin` performs as the same as `GradientBoostingClassifi` learn by looking at their log-loss on our data.

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import log_loss

custom_gbm = CustomGradientBoostingClassifier(
    n_estimators=20,
    learning_rate=0.1,
    max_depth=1
)
custom_gbm.fit(x, y)
custom_gbm_log_loss = log_loss(y, custom_gbm.predict_proba(x))
print(f"Custom GBM Log-Loss:{custom_gbm_log_loss:.15f}")

sklearn_gbm = GradientBoostingClassifier(
    n_estimators=20,
    learning_rate=0.1,
    max_depth=1
)
sklearn_gbm.fit(x, y)
sklearn_gbm_log_loss = log_loss(y, sklearn_gbm.predict_proba(x))
print(f"Scikit-learn GBM Log-Loss:{sklearn_gbm_log_loss:.15f}")
```

compare_gbm_log_loss.py hosted with ❤ by GitHub



```
# Output
Custom GBM Log-Loss:      0.46194306798
Scikit-learn GBM Log-Loss: 0.46194306798
```

As you can see in the output above, both model the same log-loss.

## Recommended Resources

In this blog post, we have reviewed all the detai boosting classification algorithm. If you are also regression algorithm, please look at the Part 1 a

**All You Need to Know about Gradient Boostin Part 1. Regression**

There are also some other great resources if you want further details of the algorithm:

- **StatQuest, Gradient Boost Part3 and Part 4** These are the YouTube videos explaining the gradient boosting classification algorithm with great visuals in a beginner-friendly way.
- **Terence Parr and Jeremy Howard, How to explain gradient boosting**While this article focuses on gradie regression instead of classification, it nicely detail of the algorithm.
- **Jerome Friedman, Greedy Function Approxi Gradient Boosting Machine** This is the orig Friedman. While it is a little hard to underst shows the flexibility of the algorithm where generalized algorithm that can deal with an having a differentiable loss function.



You can also look at the full Python code in the or the Github link below.

**Google Colaboratory**

**Gradient Boosting Algorithm – Classification masui/gradient-boosting**

## References

- Jerome Friedman, Greedy Function Approxir Boosting Machine

- Terence Parr and Jeremy Howard, How to explain gradient boosting
- Matt Bowers, How to Build a Gradient Boosting Machine from Scratch
- Wikipedia, Gradient boosting




WRITTEN BY

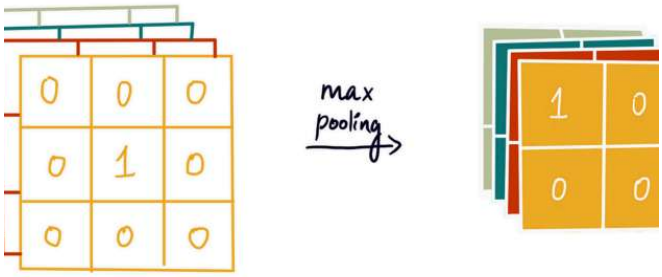
Tomonori Masui

See all from Tomonori Masui

Deep Dives | Gradient Boosting | Machine Learnir | Python

**Share This Article**

# Related Articles



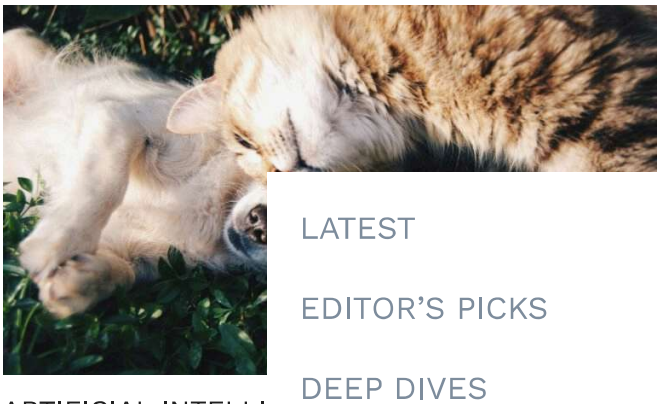

ARTIFICIAL INTELLIGENCE

## Implementing Convolutional Neural Networks in TensorFlow

Step-by-step code guide to building a Convolutional Neural Network

Shreya Rao

August 20, 2024 6 min read



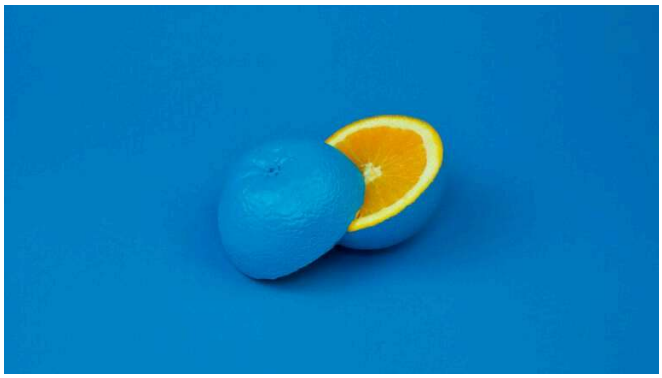
ARTIFICIAL INTELLI

## How to Forec Time Series

A beginner's guid reconciliation

Dr. Robert Kübler

August 20, 2024



DATA SCIENCE

## Hands-on Time Series Anomaly Detection using Autoencoders, with Python

Here's how to use Autoencoders to detect signals with anomalies in a few lines of...

Piero Paialunga

August 21, 2024 12 min read

MACHINE LEARNIN

## 3 AI Use Case Chatbot)

Feature engineer unstructured dat

Shaw Talebi

August 21, 2024

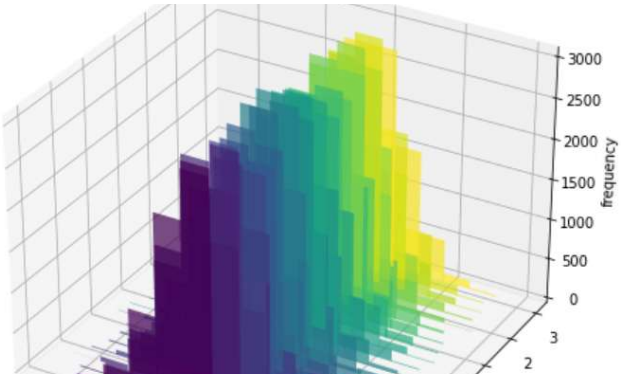
DATA SCIENCE

## Back To Basics, Part Uno: Linear Regression and Cost Function

An illustrated guide on essential machine learning concepts

Shreya Rao

February 3, 2023 6 min read




DATA SCIENCE

## Must-Know i Bivariate Nor Explained

Derivation and pr powerful concep

Luigi Battistoni

August 14, 2024 7

DATA SCIENCE

## Our Columns

Columns on TDS are carefully curated collections of posts on a particular idea or category...

TDS Editors

November 14, 2020 4 min read