The Perceptron Learning Algorithm and Convergence

Sherri Li

June 7, 2019

1 Introduction

This paper gives an overview of machine learning, illustrates how to train a perceptron model, and finally proves why the perceptron learning algorithm converges under certain conditions.

2 Artificial Intelligence

Early notions of artificial intelligence have existed for centuries. More recently, AI has been used to describe (1) machines programmed to behave like humans and (2) machines with human-like cognitive systems. The definition of AI has evolved over time. The definition became formalized in 1956, by Allen Newell and Herbert Simon during a conference at Dartmouth College. Newell and Simon presented a workshop on a Logic Theorist program and they met Marvin Minsky, who was building a simulation project for RAND Corporation during the time.[1] By the 1960s, various PhD students at MIT, Stanford, and Yale had all taken interest in AI programs that could recognize patterns in data. This leads us to today, where we use AI as an umbrella term to describe any machine that behaves in a similar to how a human would.

In 1961, George Devnol developed Unimate, an industrial robot that welded car parts in a General Motors assembly line. Unimate was a non-sentient AI designed for monotonous and dangerous work. Another example of AI is i-Robot's autonomous robot vacuum, the Roomba. It can detect obstacles in any floor as it cleans. In the past decade, AI has expanded into the field of chatbots and personal assistants. These include Apple's SIRI and Microsoft's Cortana. In 2016, Hanson Robotics even created a humanoid robot named Sophia, that Steve Kovach interviewed on an episode of Tech Insider.[2] People who design new AI must be aware of how much it costs, which communities will have access, and which communities it aims to serve.

There are countless situations humans must respond to, and the code behind an AI program would run infinitely long if programmers had to code every single possible response. The future of AI lies in training a machine to respond to more and more situations, without increasing the amount of code or programming.

[1] Lambert, Bruce. *Allen Newell, 65; Scientist Founded A Computing Field*. New York Times, 1992.

[2] Watch the video here - https://www.youtube.com/watch?v=78-1MlkxyqI.

3 Machine Learning

Machine learning, a subset of AI, was coined by electrical engineer Arthur Samuel in 1959. During that time, Samuel was building the first checkers program for the IBM 701.[3]

Today, people generally use the phrase 'machine learning' to refer to an algorithm that adapts as it sees more data. Ideally, the algorithm's performance improves after many adaptations, but these algorithms tend to rely heavily on the data they were trained upon. An algorithm is a function whose input is a set of data, and whose output is a label for each point in the set. Throughout this paper, the terms **algorithm** and **model** are used to mean the same thing. The terms **learning** and **training** are also the same.

3.1 Foundations

Machine learning was formed by taking the union of mathematical, statistical, and computer science theory. Learning algorithms aim to uncover patterns in data, and use these patterns to make predictions on new datasets.

Historically, science fiction literature and technology have pushed the boundaries of what it means to be human. Cognitive scientists and neuroscientists also play large roles in the development of new AI and ML algorithms. In the 1890s, Camillo Golgi invented a microscope staining procedure that could reveal the structure of individual neurons. Santiago Ramn y Cajal discovered growth cones on the tip of the neuron's axon, building upon Golgi's work. Cajal published illustrations of the rodent hippocampus in 1911.[4] His illustrations highlighted the resemblance between the neuron and the branches of a tree. Following Cajal's work, in 1943 Warren McCulloch and Walter Pitts published the first model of a human neuron using electrical circuits. Finally, the McCulloch-Pitts model inspired Frank Rosenblatt's formulation of the perceptron learning algorithm in 1958.

The philosophy behind training a machine arguably comes from the idea behind teaching a child. John Locke states that a child's mind begins empty, a 'blank slate.' Take the example of kids learning new words. A child learns the word 'cat,' and associates the word with any furry, four-legged creature. By this association, the child would classify dogs, lions, and horses as 'cats' — until a more well-versed person teaches the child new terms like 'dog,' 'lion,' and 'horse.' Similarly, a machine learning model will find patterns, and label data based on these patterns. The human's role is to help it learn patterns by correcting its mistakes. We call this kind of learning **supervised learning**. This paper does not cover unsupervised learning algorithms, but you may want to read more about unsupervised 'clustering' methods.

3.2 Applications

Computer vision is one application of machine learning. One such company working on computer vision and object detection is Uber. In 2018, one of their driver-less cars hit and killed a woman crossing the street in Arizona. Advocates for technology must weigh the likelihoods of human-caused accidents and machine-

[3] *The IBM 700 Series*. See https://www.ibm.com/ibm/history/ibm100/us/en/icons/ibm700series/impacts/.

[4] Cajal, Santiago. *Histology of the Nervous System of Man and Vertebrae*. Oxford University Press, 1995.

caused accidents to determine if machines will be able to prevent more deaths.[5] And as technology advances, the law must adapt to protect the rights of citizens. There should be strict regulations on driver-less vehicles, and their algorithms must be rigorously tested to ensure they can detect pedestrians and recognize road signs.

For an object detection algorithm, the input data would be pictures of various signs, taken under different lighting and different angles, each picture assigned a corresponding **label**. Label and **class** are used interchangeably in machine learning. The output data would be a set of discrete labels for each input, where labels could be the set {stop sign, not a stop sign} or the set {pedestrian, not a pedestrian}. When labels are discrete, we call it a **classification algorithm**. When labels are continuous values on some subset of $\mathbb{R}$, we call it a **regression algorithm**.

3.3 Generalizability, the goal of learning.

A successful machine learning model is one that learns to make accurate predictions on not only the train data but also the new data that it has never seen before. This desired behavior is called **generalizability**.

Creating a model that can generalize well on the test data will require extracting relevant features. **Relevant features** are variables that are strongly predictive of the label of a particular data point. If we consider spam text classification, some features that may help indicate whether a text message is SPAM or NOT SPAM could include:
(1) the number of all-caps words in the message, (2) the presence of the word "free" or "money" in the message, and (3) maybe even the length of the message.

We want the machine to detect patterns in these feature values, and adjust its algorithm to favor the features that most strongly correlated with the label. Extracting relevant features is more useful than using a complex algorithm. The following sections detail the *learning* process — feature extraction, training, and testing.

3.4 Feature Extraction

Feature extraction means representing an input instance as a vector of features. Suppose we are given four labeled text messages (assuming you or someone else has already labeled them manually):

- **1** NOT SPAM. "Help me reach my goal of raising $500 for my high school track and field fundraiser! Thank you for your support!"
- **2** SPAM. "Hope you are doing well today. If interested in a contractor position with my startup company, send me your WORD resume, BIRTHDAY, SSN, and ADDRESS. Technical Recruiter. My cell is 808-326-9607."
- **3** NOT SPAM. "Your Bank of America credit card payment of $256.08 is due by 4/1/2019. Please sign in view bill and pay."
- **4** SPAM. "WINNER! You have been selected to receive a $999 prize reward! To claim call 0907146174. Valid 24 hours only."

[5] Zaveri, Mihir. *Prosecutors Don't Plan to Charge Uber in Self-Driving Cars Fatal Accident*. New York Times, 2019.

Suppose we choose the follow features to extract from each message:
(x_1, x_2, x_3) = (ratio of all-capitalized words to all-lowercase words, ratio of numbers to words, number of exclamation marks)

Then we can express each text messages as a 3-tuple in the table below. *Note: Avoid using (x, y, z) to represent the feature vector, because y is commonly used to denote label. So we say $\vec{x} = (x_1, \ldots, x_{dim})$.* Here, $dim = 3$.

	x_1 **all-caps/all-lowercase**	x_2 **numbers/words**	x_3 **# exclamations**	**label**
1	0	3/20	2	not spam
2	4/26	10/30	0	spam
3	0	11/18	0	not spam
4	1/15	15/16	2	spam

Another way to visualize these train data is to graph the instances by their features. In the graph below, blue points are spam and red points are not spam.[6]

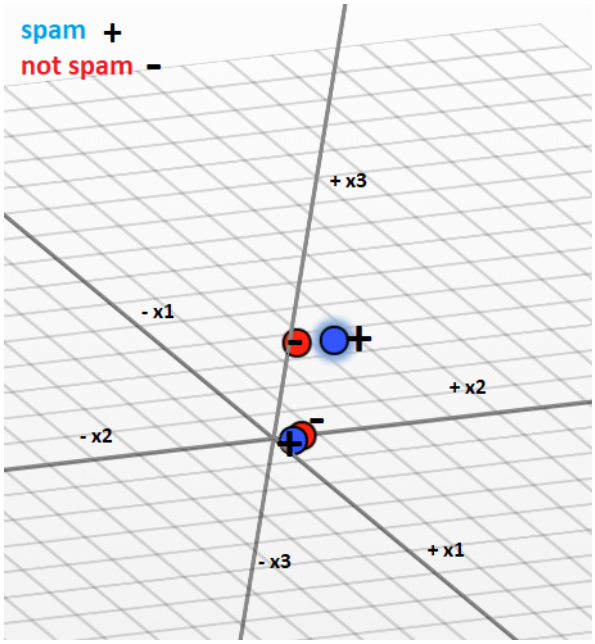


Figure 1: Train instances plotted on x_1-x_2-x_3 axis.

By inspection, it doesn't look like we can easily find a plane that correctly separates the data into two classes (spam, not spam). If you use the plane $x_3 = 1$ for example, there would be both **-** and **+** points on the same side of the plane.

Let's try ditching x_3, the number of exclamations feature. If we keep only x_1 and x_2, then our feature dimension drops from 3 to 2.

[6]Plot created using CPM 3D Plot software.

Now we can plot the instances $(0, 0.15, notspam)$, $(0.15, 0.33, spam)$, $(0, 0.61, notspam)$, $(0.07, 0.94, spam)$ in 2-space below.[7]

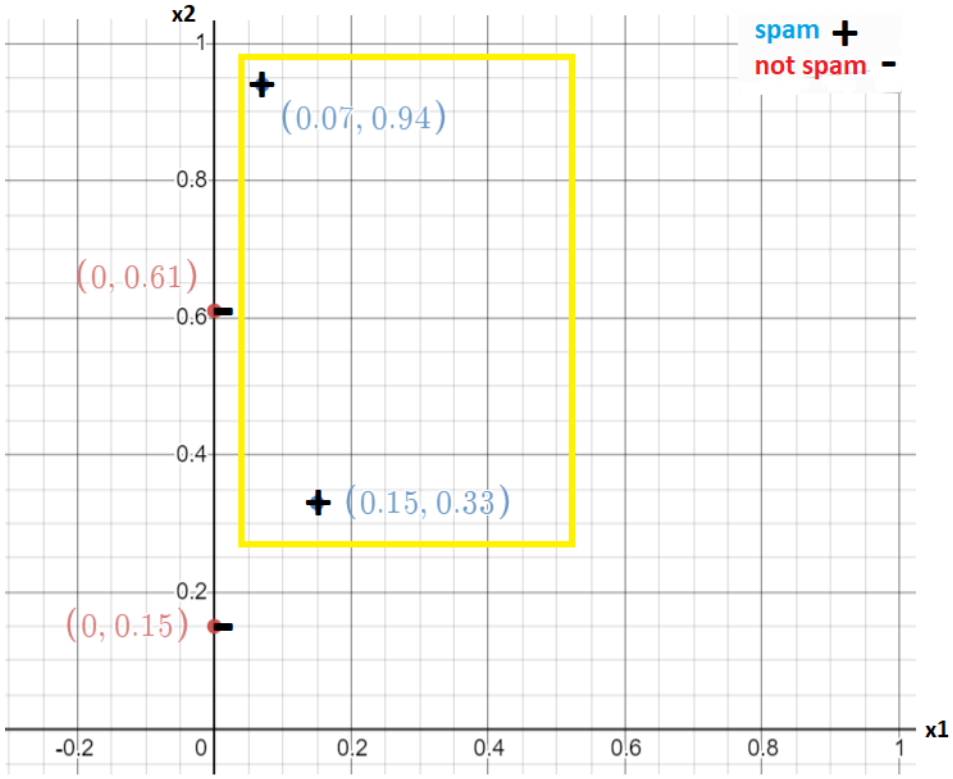


Figure 2: Train instances plotted on x_1-x_2 axis.

We can draw a rectangular boundary to separate the *spam* and *not spam* data points. This boundary was easier to spot in 2-space than in 3-space. The next section will describe the training phase.

3.5 Training

The train phase involves learning a **hypothesis** AKA **classifier** — a set of rules or functions that tell us how to classify an input as *SPAM* or *NOT SPAM*.

We can define a hypothesis $g_1(x)$ using the corners of the rectangle as our parameters:

$$g_1(x) = \begin{cases} SPAM \text{ if } 0.1 < x_1 < 0.5 \text{ and } 0.3 < x_2 < 1 \\ NOTSPAM \text{ otherwise} \end{cases}$$

[7] Plot created using Desmos software.

During training, the model learns which features were most predictive of the label. It makes informed adjustments to the model, based on calculating the error between the its predicted label $\hat{y}$ and the actual label y. The model learns better when it sees more diverse data. Suppose our training set had more points, such as $(0.5, 0.5, spam), (0.3, 0.7, spam), (0.3, 0.1, notspam), (0.6, 0.9, notspam)$. See the new graph below.

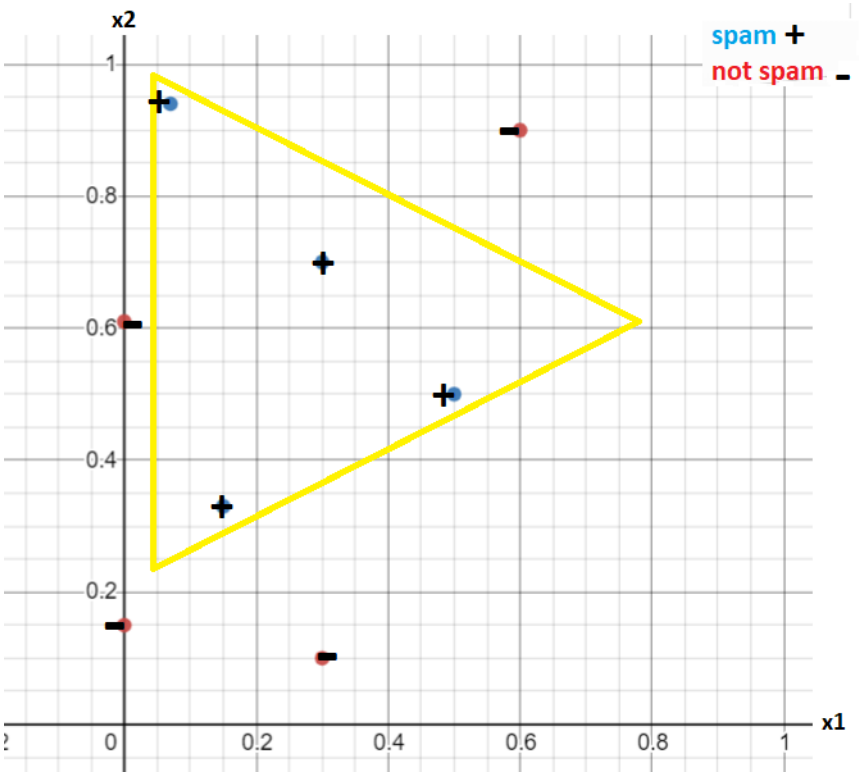


Figure 3: Plot additional train instances, g(x).

With the addition of these train instances, we might tweak our model to predict $spam$ if it lies within the yellow triangle boundaries. We can define a new $g_2(x)$ to reflect the three lines that define the triangle. It may look roughly like:

$g_2(x) =$

$$\begin{cases} SPAM \text{ if } 0.1 < x_1 < 0.8 \text{ and } 0.5x_1 + 0.2 < x_2 < -0.57x_1 + 1 \\ NOTSPAM \text{ otherwise} \end{cases}$$

We could also decide keep the old rectangle and just shift the boundaries up or down, left or right. You, the scientist, have the power to create and test your boundaries against different test sets, to see which $g(x)$ predicted with the highest accuracy.

One can use a **loss function** $L(y, \hat{y})$ (aka error function, aka objective function) to train a model to learn good boundaries on x_1 and x_2. Boundaries are also called **parameters**. Regardless of your jargon, the function $L(y, \hat{y})$ will measure how far off your model's prediction was from the actual label. Then, the model

will update its parameters in a way that will minimize the value of L.

3.6 Cross-Validation

You are given the labels of your train set, but never the labels of your test set. It's your model's job to predict the test labels. This means you can't ever be sure how accurate your test predictions are. So how can we get a good estimate on how accurate our model will be?

K-fold cross validation is a commonly used method to predict test accuracy. It's used during training to find the highest-performing model among a set of different models. There is some set theory involved!
First, we 'hold out' a portion of our labeled train set D. We treat this 'held out' set as a 'test' set. But because it's labeled, we can verify our accuracy.
Then, we partition the remaining train set D into k equal subsets, $D = D_1 \cup D_2 \cup \cdots \cup D_k$. Then, partition each subset D_i into k equal sub-subsets, $D_i = D_{i_1} \cup \cdots \cup D_{i_k}$. The diagram below illustrates this idea.[8]

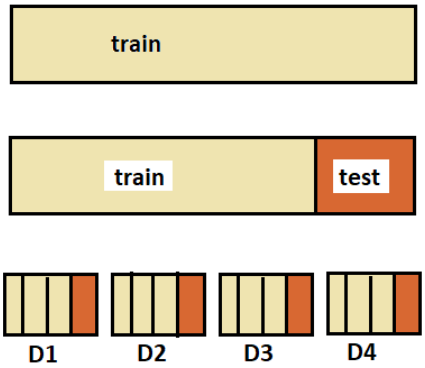


Figure 4: k-fold CV with $k = 4$.

We would then train each classifier i on $k-1$ of the D_{i_i} sub-subsets. Then we can test each classifier on the k^{th} sub-subset D_{i_k} ($i \in [1, 4], k = 4$), and check its predicted labels against the actual labels. Finally, we pick a classifier with a high test accuracy.

If interested, read more on the other flavors of CV, including Leave-One-Out CV, Leave-P-Out CV, and repeated random sub-sampling validation.

3.7 Testing

Finally, we **test** our model. This entails using our classifier, which we've built during the train phase, to predict the label of a never-before-seen test instance. It's essential that test instances come from the same population as train instances. The assumption in ML is that all training and test instances are **iid** — independent and identically distributed. (This means we assume all instances have the same probability distribution

[8] Diagram created using MS Paint software.

and are mutually independent.)

You want your model to make accurate predictions on this new test data. Consider the test instance below:

- Your W-2s are ready! FILE your simple tax returns for FREE! Sign back in to TurboTax today.

We'll represent our test instance $\vec{x}_{test}$ as a feature vector $(3/14, 1/17)$:= (ratio of all-caps words to lowercase words, ratio of numbers to words).

Let's calculate $g_1(\vec{x}_{test})$ and $g_2(\vec{x}_{test})$. Note: $\vec{x}_{test} = (3/14, 1/17) = (0.21, 0.06)$.
g_1 says $\vec{x}_{test}$ is spam, if $0.1 < \vec{x}_{test_1} < 0.5$ and $0.3 < \vec{x}_{test_2} < 1$.
Thus, $g_1(\vec{x}_{test}) = notspam$.
g_2 says $\vec{x}_{test}$ is spam, if $0.1 < \vec{x}_{test_1} < 0.5$ and $0.5(0.21) + 0.2 < x_{test_2} < -0.57(0.21) + 1$.
Thus, $g_2(\vec{x}_{test}) = notspam$.

Both models predict that $\vec{x}_{test}$ is not a spam text. That's because $\vec{x}_{test}$ outside of both the rectangle and triangle. However, if we took some point $\vec{x}_{test'} \in \Delta$, $\vec{x}_{test'} \notin \square$:

we'll get $g_2(\vec{x}_{test'}) = spam$ and $g_1(\vec{x}_{test'}) = notspam$.

What if we extracted different features, *(has all-caps, has numbers, has exclamations)*, instead? And what if we made our hypothesis $g(\vec{x}) = spam$ if $\vec{x} = (yes, yes, yes)$ (otherwise, not spam)? Then $\hat{y} = g(\vec{x_{test}}) = spam$, a mistake. It would, however, correctly label three out of the four original train instances. An **overfit** model has high train accuracy and low test accuracy — it is tuned too closely to the train set, and detects patterns that are just 'noise.' Likewise, a model not complex enough to capture underlying patterns in the train set is an **underfit** model.[9]

4 Perceptron Learning Algorithm

4.1 Originations

Perceptron is an **online classification algorithm** that *learns* a **linear decision boundary**. Essentially, it's a function that takes a set of labeled data as input, returns a linear boundary as output. In general we refer to the line as a **hyper-plane**. In 2-D space, the hyper-plane is just a line.

Below we define certain properties of perceptron:

1. It is an **online** algorithm — training data is given to the model in sequential order, as a stream of incoming data. The model adjusts itself after each datapoint it sees. It makes these adjustments based on how different its prediction was from the actual label.

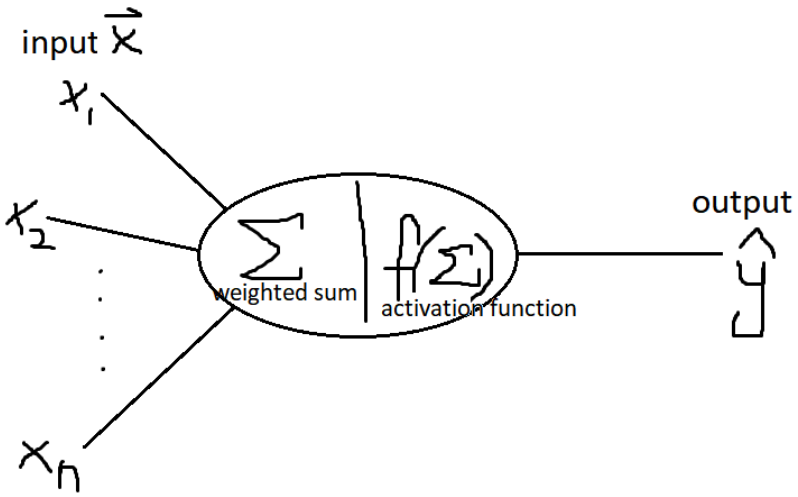
2. It is an **error-driven** algorithm — the algorithm updates itself only when it makes a prediction error at train time.

[9] See reference [4] for ways to prevent underfit and overfit.

3. It falls under the category of **classification** algorithms — the algorithm classifies the testing data into one of two discrete categories: {label X, NOT label X}.

4. **Linear boundary** — the hyper-plane separates the data points in a D-dimensional space into two spaces, $+$ and $-$.

The illustration below visualizes perceptron as a neural network. A **neural network** is a mathematical model based on the biological neuron. There is single layer of computation necessary to map the input to the output label. Adding more 'layers' equates to making more adjustments to the inputs before arriving at the output. In practice, data is complex, and multiple layers are required.[10]



The perceptron algorithm transforms $\vec{x}$, with n dimensions, to some label $\hat{y} \in \{-1, +1\}$. These n components represent the dendrites of the neuron. The cell body and nucleus are composed of a weighted sum and processing this sum to the output. The next section describes these steps in detail.

4.2 Vanilla Perceptron

Inside the cell body, perceptron calculates a weighted sum $S = \sum_{i=1}^{n} w_i x_i$ of each input, and adjusts w_i using an algorithm which we will prove terminates under certain conditions. In practice, the values of in $\vec{w}$ are initialized to very small values.

Including a **bias term** b as an input is useful when you want to account for an **unbalanced dataset**, a set of data where there is a majority of data points that belong to one class over the other class. This bias term b horizontally shifts the output hyper-plane. When using b, the sum becomes $S = \sum_{i=1}^{n} w_i x_i + b$. Think of the hyper-plane as a line $y = \vec{w} \cdot \vec{x} + b$. Adjustments made to the weights change the slope of the hyper-plane, while adjustments to the bias will shift it.

Second, perceptron uses an **activation function** (aka transfer function), $f : \mathbb{R} \to \{-1, +1\}$. $Spam$ can correspond to $+1$, while $not - spam$ can correspond to -1. Perceptron passes the previously calculated

[10] To learn more about hidden layers in neural networks, see reference [9], Chapter 6, *Deep Feedforward Networks*.

sum S into the activation function, and ultimately the predicted label $\hat{y} = f(S)$.

For example $f(S) =$:

$$\begin{cases} +1 & S > 0 \\ -1 & S \leq 0 \end{cases}$$

The activation function is typically a piece-wise step function, sigmoid function, or a hyperbolic tangent function.

4.3 Extension: One vs All

Sometimes we may want to predict data that can fall into more than two labels. Let's consider we are building a model to support the vision of a self-driving car. Our inputs are images of various traffic lights, and we want our model to predict which color bulb is lit based on the image. In this case, we'll feed the model pictures of traffic lights under different weather conditions and from different angles, and we want it to tell us if the light was {red, yellow, green}.

First, we group the classifiers $\binom{3}{2}$ times to get {red, yellow or green}, {yellow, red or green}, and {green, yellow or red}. We will have 3 functions f_R, f_B, f_G, each mapping $\mathbb{R}_n$ to $\mathbb{R}$.
For each input x_i, the model assigns its label y_i as $argmax_c(f_c(x))$, the largest of the values $f_R(x_i)$, $f_B(x_i)$, $f_G(x_i)$. [11]

5 Perceptron Pseudocode

Rosenblatt's perceptron learning algorithm was first simulated on an IBM 704 machine in Cornell's Aeronautical Laboratory. Much early research in perceptrons was aimed to train machines for image recognition, and funded by the the United States Office of Naval Research.

Recall the diagram in Section 4.1, let's call the set of all inputs $(\vec{x}, y)$ the set **D**. For each instance $(\vec{x}, y) \in \mathbf{D}$, $\vec{x}$ is a feature vector $(x_1, \ldots, x_{dim})$ with $x_i \in \mathbb{R}$. y is its label, where $y \in \{+1, -1\}$.

The algorithm below trains a line to classify the points in **D**. It can be implemented in any programming language, but certain languages like Python include built-in algorithms that you can import and use.[12]

```
1: function PERCEPTRONTRAIN(D, dim)
2:     weights ← [random(0, 1)] * dim ▷ Initialize all weights to some random value in between 0 and 1
3:     bias ← 0                                                    ▷ Initialize bias term to 0
4:     while TRUE do
5:         mistakes ← 0                                  ▷ Count number of mislabels made during each run
6:         for each (x⃗, y) in D do
7:             sum ← 0
8:             for j in 1, ..., dim do
9:                 sum ← sum + weights_j · x_j   ▷ Calculate weighted sum of features for each instance
```

[11] To learn about multi-class classification, see reference [2], Chapter 6, *Beyond Binary Classification*.
[12] See reference [6] for another implementation of PerceptronTrain().

```
            end for
            S ← sum + bias
            ŷ ← f(S)                                   ▷ ŷ is predicted label
            if ŷ · y ≤ 0 then                          ▷ Predicted label differs from actual label
                weights ← weights + Δweights           ▷ Update weights vector
                bias ← bias + Δbias                    ▷ Define Δ below! *
                mistakes ← mistakes + 1                ▷ Increment mistake count
            end if
        end for
        if mistakes = 0 then                           ▷ Model made no mistake on any training instance
            break                                      ▷ Break out of the algorithm
        end if                                         ▷ Otherwise, continue looping
    end while
    return (weights, bias)
end function
```

The training algorithm outputs a weights vector and a bias term that define some line $h = \vec{w} \cdot \vec{x} + b$. We call this line our 'hyper-plane,' and it classifies a point $(\vec{x}, y)$ as either $+1$ or -1. In practice, one can increase the threshold $mistakes$ from 0 up to 30% of your dataset **D**. Allowing more room for error during training helps prevent overfit.

* We want $\Delta weights_i$ and $\Delta bias$ to represent the optimal change in slope and position of the hyper-plane. We find them through a process known as gradient descent.[13]

Define $error\ E$ as the difference between the actual and predicted value.

$$E = |y - \hat{y}| = |y - \vec{w} \cdot \vec{x} + b|$$

Set $\Delta w_i = \frac{\partial E}{\partial w_i} = \vec{x}$. Similarly, $\Delta b = \frac{\partial E}{\partial b} = 1$.
Recall that $\hat{y}, y \in \{+1, -1\}$. Because of the absolute value, we can rewrite $\Delta w_i = x_i$ as $y_i \vec{x}$. We use this expression for the proof of convergence in Section 6.

5.1 Testing Algorithm

After training the perceptron, you will probably want to use it to make predictions on a test set. The PerceptronTest() algorithm below takes the output of PerceptronTrain() as input. It also takes some test instance $\vec{x}' = (x'_1, x'_2, \ldots, x'_{dim})$ as input. Its output is $\hat{y}$, the predicted label of $\vec{x}'$.

```
function PERCEPTRONTEST(weights, bias, x⃗′)
    S ← Σ_{i=1}^{dim} weights_i x′_i + b
    ŷ ← f(S)
    return ŷ                                           ▷ Return the predicted label
end function
```

[13] For details on gradient descent, see reference [4].

Note on terminology — perceptron is called a 'feed-forward neural network with backward propagation.' Inputs are passed forward, while errors are propagated back and used to update the model.

5.2 Voted Perceptron

The order you feed your perceptron model train instances is important, because the model only stores the most recent hyperplane in memory. It forgets older, and possibly more stable, hyperplanes. 'Voted perceptron' is an extension of vanilla perceptron, that saves all hyperplanes used and how long each 'survived' before it was updated. Then, each hyperplane votes on the predicted class of the test instance. A hyperplane that 'survived' longer will have more votes.[14]

6 Convergence of Perceptron

In this section, we discuss the conditions under which the PerceptronTrain() algorithm converges — line (18)'s **if condition** is eventually satisfied, which in turn terminates the **while loop** in line (4).

6.1 Linear Separability

A dataset **D** is **linearly separable** if there exists a hyper-plane that classifies all points $(\vec{x}, y) \in \mathbf{D}$ correctly. Let's consider labels $y \in \{+, -\}$ in Figure 5 below.[15] Such a hyper-plane exists.

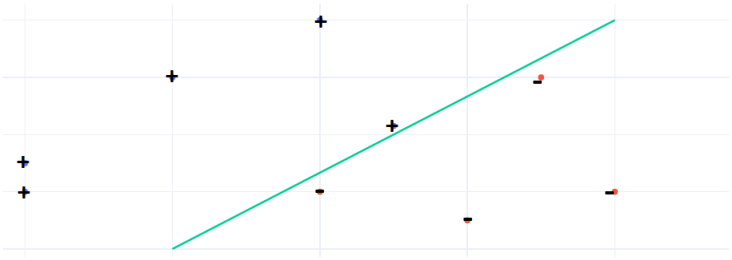


Figure 5: Linearly separable dataset.

On the other hand, the datapoints in Figure 6 cannot be separated into **+** and **-** points. The algorithm would never converge on this dataset, the updates to $\vec{w}$ would eventually just fluctuate and repeat themselves, we call this "getting stuck" in an infinite loop.

6.2 Margin

The **margin** γ of a linearly separable dataset is the distance between separating hyper-plane and the point $(\vec{x}, y) \in \mathbf{D}$ closest to it. We assume points do not lie on the hyper-plane, so it follows that $\gamma > 0$.

[14] To learn about other flavors of perceptron, see reference [2].

[15] Plot made using Plotly Dash software.

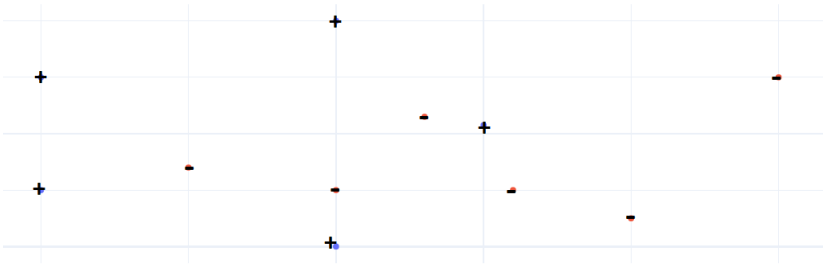


Figure 6: Non linearly separable dataset.

6.3 Theorem

Suppose the following conditions hold:

1. $\mathbf{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is a linearly separable dataset with margin $\gamma > 0$.
2. The distance from each point in the dataset to the origin is bound by some $R \in \mathbb{R}$: $\forall x \in D, ||x|| \leq R$.

Under (1) and (2), the perceptron algorithm terminates after at most $\frac{R^2}{\gamma^2}$ mistakes.

6.4 Proof

For simplicity, consider no bias term in this proof.
Then, our hyperplane h is defined by its normal vector $\vec{w}$. Consider the k^{th} mis-prediction that the algorithm makes during training, and suppose it mislabels train instance $(\vec{x_l}, y_l)$. Then, we will derive bounds on the magnitude $||w_k||$ and eventually conclude that $k \leq \frac{R^2}{\gamma^2}$. We will refer to the pseudocode PerceptronTrain() algorithm throughout the proof.

Lower Bound
By assumption, k was a mistake. By line (14) of PerceptronTrain(), weight w_{k+1} gets updated:
$w_{k+1} = w_k + y_l \vec{x_l}$ ★
By condition 1 — linear separability — there exists a hyper-plane that correctly labels every point in **D**. It follows that this hyper-plane has some weight vector $\vec{\mathbf{w}}$ such that $(\hat{y})y_i = (\vec{\mathbf{w}}\vec{x_i})y_i \geq \gamma$ for all $(\vec{x_i}, y_i) \in \mathbf{D}$. Let's make $\vec{\mathbf{w}}$ a unit vector for the purposes of our proof. This means all points are at least γ distance from the plane.
Dot product both sides of ★ by $\vec{\mathbf{w}}$, and simplify.
Note. Dot products are commutative and associative.

$$\begin{aligned} \vec{w_{k+1}}\vec{\mathbf{w}} &= (w_k + y_l \vec{x_l})\vec{\mathbf{w}} \\ &= \vec{w_k}\vec{\mathbf{w}} + y_l \vec{x_l}\vec{\mathbf{w}} \\ &= \vec{w_k}\vec{\mathbf{w}} + \vec{\mathbf{w}}\vec{x_l} y_l \\ &= \vec{w_k}\vec{\mathbf{w}} + (\vec{\mathbf{w}}\vec{x_l}) y_l \\ &\geq \vec{w_k}\vec{\mathbf{w}} + \gamma \end{aligned}$$

Note. Vector dot products are less than or equal to the product of vector magnitudes.

$$||\vec{w_{k+1}}|| \text{ x } ||\vec{\mathbf{w}}|| \geq \vec{w_{k+1}} \cdot \vec{\mathbf{w}} \geq \vec{w_k} \cdot \vec{\mathbf{w}} + \gamma$$

By definition of a unit normal vector, $||\vec{\mathbf{w}}|| = 1$. Use this fact below:

$$||\vec{w_{k+1}}|| = ||\vec{w_{k+1}}|| \text{ x } ||\vec{\mathbf{w}}|| \geq \vec{w_{k+1}} \cdot \vec{\mathbf{w}} \geq \vec{w_k} \cdot \vec{\mathbf{w}} + \gamma$$
$$||\vec{w_{k+1}}|| \geq \vec{w_k} \cdot \vec{\mathbf{w}} + \gamma \qquad \text{(A)}$$

Claim. $||\vec{w_{k+1}}|| \geq k\gamma$.

Base Case. $k = 1$.
In theory, we can assume that the first-initialized weights $\vec{w_1}$ in line (2) of PerceptronTrain() are zeros. Then apply (A) above:

$$||\vec{w_2}|| \geq \vec{w_1}\vec{\mathbf{w}} + \gamma \geq 0 \cdot \vec{\mathbf{w}} + \gamma \geq \gamma$$

Inductive Step. Suppose $\forall k, 1 \leq k \leq n$, that $||\vec{w_{k+1}}|| \geq k\gamma$. Want to show $||\vec{w_{k+2}}|| \geq (k+1)\gamma$. Using (A) with $k = k + 1$, we get

$$||\vec{w_{k+2}}|| \geq \vec{w_{k+1}}\vec{\mathbf{w}} + \gamma$$

Use the linear separability condition, $\gamma > 0$, and then apply the inductive hypothesis

$$\begin{aligned} ||\vec{w_{k+2}}|| &\geq ||\vec{w_{k+1}}\vec{\mathbf{w}}|| + \gamma \\ &\geq k\gamma + \gamma \\ &\geq (k+1)\gamma \end{aligned}$$

$\therefore$ By induction, $\forall k \in \mathbb{N}$, $||\vec{w_{k+1}}|| \geq k\gamma$. □

Upper Bound
We'll use the definitions of vector magnitude and inner product.

$$\begin{aligned} ||\vec{w_{k+1}}||^2 &= (\vec{w_{k+1}}) \cdot (\vec{w_{k+1}}) \\ &= (\vec{w_k} + y_l\vec{x_l}) \cdot (\vec{w_k} + y_l\vec{x_l}) \\ &= (\vec{w_k}) \cdot (\vec{w_k}) + 2y_l\vec{x_l} + (y_l\vec{x_l}) \cdot (y_l\vec{x_l}) \\ &= ||\vec{w_k}||^2 + 2y_l\vec{x_l} + ||(y_l\vec{x_l})||^2 \end{aligned}$$

Use the fact that k was a mistake, meaning $y_l\vec{x_l} < 0$. Use also the fact that $y \in \{+1, -1\}$. Furthermore, condition 2 tells us all $||x||$ are bounded from above by R.

$$\begin{aligned} ||\vec{w_{k+1}}||^2 &\leq ||\vec{w_k}||^2 + 0 + ||1 \cdot \vec{x_l}||^2 \\ &\leq ||\vec{w_k}||^2 + R^2 \qquad \text{(B)} \end{aligned}$$

Claim. $||\vec{w_{k+1}}||^2 \leq kR^2$.

Base Case. $k = 1$.
$||\vec{w_2}||^2 \leq ||\vec{w_1}||^2 + R^2$
By line (2) of PerceptronTrain(), weights are initialized to a small value very close to 0. For theoretical purposes, assume the initial weight $\vec{w_1}$ is zero. Then, $||\vec{w_2}||^2 \leq ||\vec{w_1}||^2 + R^2 \leq 0 + R^2 \leq R^2$.

Inductive Step. Suppose $\forall k, 1 \leq k \leq n$, that $||\vec{w_{k+1}}||^2 \leq kR^2$. Want to show $||\vec{w_{k+2}}||^2 \leq (k+1)R^2$.
Using (B) with $k = k + 1$, we get

$$||\vec{w_{k+2}}||^2 \leq ||\vec{w_{k+1}}||^2 + R^2$$

Apply the inductive hypothesis

$$\begin{aligned} ||\vec{w_{k+1}}||^2 + R^2 &\leq kR^2 + R^2 \\ &\leq (k+1)R^2 \end{aligned}$$

$\therefore$ By induction, $\forall k \in \mathbb{N}$,$||\vec{w_{k+1}}||^2 \leq kR^2$. □

Combine Bounds
Upper bound is

$$||\vec{w_{k+1}}||^2 \leq kR^2$$

Square the lower bound to get

$$||\vec{w_{k+1}}||^2 \geq k^2\gamma^2$$

Combine the two inequalities

$$k^2\gamma^2 \leq ||\vec{w_{k+1}}||^2 \leq kR^2$$

By assumption, both $k > 0$ and $\gamma > 0$. This allows us to divide both sides by k and by γ^2.

$$\begin{aligned} k^2\gamma^2 &\leq kR^2 \\ k\gamma^2 &\leq R^2 \\ k &\leq \frac{R^2}{\gamma^2} \end{aligned}$$

Thus, k is finite and is bound from above by $\frac{R^2}{\gamma^2}$. This result tells us that the PerceptronTrain() algorithm terminates after making at most some $\frac{R^2}{\gamma^2}$ prediction errors. As the margin increases, the algorithm converges faster. □

7 References

[1] Rosenblatt, Frank. (1962). *Principles of Neurodynamics*. Spartan Books, 1962.

[2] Daume, Hal. *A Course in Machine Learning*. Self-published, 2017. Online textbook, see http://ciml.info/.

[3] Weinberger, Kilian. *Curse of Dimensionality and Perceptron*. Cornell CS4780, 2018. Online lecture notes, see http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html.

[4] Ng, Andrew. *CS229 Lecture notes*. Stanford CS229, 2018. Online lecture notes, see http://cs229.stanford.edu/notes/cs229-notes1.pdf.

[5] Collins, Michael. *Convergence Proof for the Perceptron Algorithm*. Columbia COMS E6998 Machine Learning for Natural Language Processing, 2012.

[6] Brownlee, Jason. *How To Implement The Perceptron Algorithm From Scratch In Python*. Machine Learning Mastery, 2016.

[7] Marr, Bernard. *What Is The Difference Between Artificial Intelligence And Machine Learning?*. Forbes, 2016.

[8] Shiffman, Daniel. *The Coding Train*. Self-published website, 2017.

[9] Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press, 2016. Online textbook, see http://www.deeplearningbook.org.