Be Greedy, but Not Too Greedy: Mistake Bounds for a New Perceptron Algorithm

Benjamin Steiner
Supervised by Dr. Tuğkan Batu

A Dissertation submitted to the Department of Mathematics of the London School of Economics and Political Science

March 24, 2023

Summary

In this dissertation, we discuss the Perceptron Algorithm, its pros and cons, and modifications to tackle some of the issues with it. Firstly, we discuss the Perceptron and understand why and how it works. We explore this by considering the update rule in two ways: firstly through the inner product, and then geometrically. We present the main topic of this dissertation in Chapter 3, the Greedy Perceptron. We first give the framework and then analyse its various properties such as the new update rule and mistake bound. We also give a few examples comparing it to the standard Perceptron. The code for the comparison shown in Figure 3.4 is given in the Appendix.

A limitation of the Perceptron is the requirement for the data to be linearly separable. To reconcile this, in Chapter 4 we show the Fuzzy Perceptron, introduced in Keller and Hunt (1985). It is one method to deal with linearly inseparable datasets. It achieves this by "ignoring" some of the data that is almost equidistant from the cluster centres. Chapter 5 then considers an alternative way to handle this case: the Margin-Fuzzy Perceptron. Rather than ignoring the data before the start of the algorithm, it creates a margin at each iteration that hopefully contains all of the overlapping data.

The most important result is Theorem 9 which gives an upper bound on the number of mistakes that the Perceptron makes: $(\frac{R}{\gamma})^2$. This mistake bound will be the overarching theme between chapters and will allow us to compare the subsequent variants we introduce to each other. We prove for the Greedy Perceptron Algorithm that, while in certain cases it is able to find a solution in fewer iterations, it comes at a cost of a worse mistake bound: $(\frac{\eta_{max}}{\eta_{min}})^2 \cdot (\frac{R}{\gamma})^2$.

We are also able to compare the mistake bounds of the two ideas given in Chapters 4 and 5. However, Theorems 23 and 25 give a different way to compare the Fuzzy Perceptron and Margin-Fuzzy Perceptron. Rather than only considering the mistake bounds for these variants, we are able to compare them to each other by examining how large the gap is between the clusters.

Contents

Chapter 1

Introduction

1.1 Historical Context

In McCulloch and Pitts (1943), they suggested that we can take inspiration from the human brain to solve classification problems. To this end, they built the first mathematical model of a neuron. Then, Rosenblatt (1958) developed a new model that was instrumental in the development of Artifical Intelligence (AI). While most neural networks no longer use the Perceptron neuron model, it was the building blocks of Artificial Neural Networks and provided the framework for further research. A more common approach nowadays for gradient methods uses the Backpropagation model suggested in Werbos (1974).

The Perceptron Algorithm was the first example of a binary classification algorithm. By which we mean that we aim to find a hyperplane that separates our two classes of data - with one class on one side, and the other class on the opposite side. We call this hyerplane a *decision boundary*. The Perceptron algorithm uses the most simpole of them all - a linear decision boundary.

The Perceptron Algorithm falls under the class of *Online Machine Learning* algorithms. These types of algorithms learn by using one data point at a time to update the decision boundary, as opposed to using mini-batches or the entire data set at each iteration. Online learning is especially useful in situations with large amounts of data where it is computationally infeasible to use a large proportion of the data set at each iteration to learn the boundary.

1.2 Structure

In Chapter 2, we show the Perceptron Algorithm and build the knowledge required to critique some of the flaws with it. Chapter 3 introduces a variant we call the Greedy Perceptron Algorithm, which addresses a fundamental question with the standard Perceptron update rule. Chapter 4 introduces the Fuzzy Perceptron: a way to deal

with the case where the data is not linearly separable - more akin to many problems in the real world. Finally, Chapter 5 is a re-formulation of the Fuzzy Perceptron. Motivated by the Soft-Margin Support Vector Machine (SVM) first presented in Cortes and Vapnik (1995), it shows a different way to deal with the problematic overlapping data.

1.3 Original Contributions

Throughout, we provide new theorems and re-formulations of the Perceptron Algorithm. To the best of my knowledge, Theorems 8, 15, 16, 17, 18, 22, 23, 24, and 25 are all new proofs.

In Chapter 2, Theorem 8 is a new alternative proof to Theorem 7. Rather than considering the effect of the update to $y\langle w_k, x\rangle$, we show how this is equivalent to examining how the angle between the normal and feature vector changes.

In Chaper 3, we introduce a new variant called the Greedy Perceptron Algorithm. Theorems 15 and 16 use the ideas for the Greedy Perceptron applied to the structure of the proof for the mistake bound of the standard Perceptron: Theorem 9. Theorem 17 is a corollary of the previous theorem and is simply an observation. The consequence of Theorems 15 and 16 is that we can now derive a new upper bound for the number of mistakes the Greedy Perceptron makes which we show in Theorem 18.

In Chapter 4, we give the mistake bound of the Fuzzy Perceptron, Theorem 22, for completeness. A stronger result we give is Theorem 23: that the Fuzzy Perceptron creates a range of angles that any hyperplane in this range would separate the data. The purpose of this geometric interpretation is that it allows us to compare the Fuzzy Perceptron with the Margin-Fuzzy Perceptron introduced in Chapter 5.

In Chapter 5 we show the Margin-Fuzzy Perceptron: an alternative to the Fuzzy Perceptron that also deals with cases where the data is not linearly inseparable. Theorem 24 shows how this variant allows us to directly control the upper bound of number of updates it makes. Finally, Theorem 25 is the geometric interpretation of the motivation for the Margin-Fuzzy Perceptron. It is Chapter 5's equivalent proof to Theorem 23 in Chapter 4, and allows us to compare the Fuzzy Perceptron to the Margin-Fuzzy Perceptron.

1.4 Notation

Throughout this dissertation we re-use some important concepts and now define them. Firstly, we introduce the notion of the Euclidean Norm.

Definition 1. Let v be some vector, then we define the *Euclidean Norm* of v to be

$$\|\mathbf{v}\| \coloneqq \sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}.$$

With this, we can now introduce the inner product.

Definition 2. Let u and v be any two vectors, and $\|u\|$ and $\|v\|$ be the standard Euclidean Norms of u and v. We define the *inner product* between u and v to be

$$\langle u, v \rangle = \|u\| \cdot \|v\| \cdot \cos \alpha.$$

For the proofs of mistake bounds, we require two important concepts: the radius and margin of the data. We define the radius to be the maximum norm of all the data, so we have that $R \coloneqq \max_{x \in X} \|x\|$.

Recall that for a vector x and a normal to a hyperplane w with $\|w\| = 1$, $\langle w, x \rangle$ is the *signed* perpendicular distance between the boundary and the vector x. By this we mean that $\langle w, x \rangle = d$ if a vector x is on the side that the normal points to, and $-d$ if it is on the other side. We define the margin γ as the minimum distance between the boundary and any of the data. More formally, $\gamma \coloneqq \min_{x \in X} y \langle w_k, x \rangle$.

Chapter 2

The Perceptron Algorithm

2.1 Problem Setting

In this chapter, we introduce the main topic of this dissertation: The Perceptron Algorithm. The goal of the algorithm is, given a dataset where each point is classified into one of two classes, to find a weight vector, w, such that the normal to this line *separates* our two classes of data.By *separate*, we mean that we want one class *above* the line, and the other *below* the line. We can think of our classes as some characteristic of our data such as *"Male or Female"* or *"Labour or Conservative Party Supporter"*.

The algorithm uses an iterative method of updating the weight vector until a correct weight vector has been found. This corresponds to the normal to the decision boundary pointing towards the vectors belonging to Class 1. We show what this looks like in Figure 2.1. Note that in order to find a hyperplane that separates the data, the data must be *linearly separable*. We omit the proof, but it is worth adding that if the data is linearly separable, we can guarantee convergence of the algorithm. That is, we can be sure that we will find a hyperplane that separates the data. It is worth noting that while there will be many different boundaries that separate our data, the algorithm will terminate once it has found the first separating decision boundary and will not try to find a "better" one after that.

2.2 Preliminaries

Before we move onto explaining this any further, let us first define the notation we are going to use. We denote our dataset by $X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where m is the length of our dataset. We use $x_i \in \mathbb{R}^n$ to denote the i^{th} observation, and $y_i \in \{-1, 1\}$ for the class of this observation. We call w our *weight vector* and this is the normal to our decision boundary. To reduce the amount of notation, we will often just use x and its corresponding y.

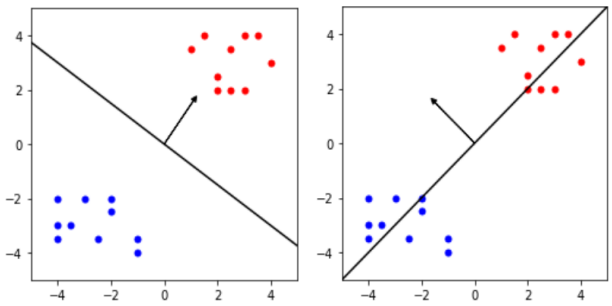

Figure 2.1: The arrow is the weight vector, and the line is the corresponding decision boundary. Left: An example decision boundary that separates our two classes. Right: An example where the boundary does not separate the two classes of data.

Definition 3 (Class of x)**.** Given a dataset X, each vector $x \in X$ belongs to one of two classes. We define the class it belongs to, y, as follows:

$$y = \begin{cases} +1 & \text{if } x \text{ belongs to Class 1} \\ -1 & \text{if } x \text{ belongs to Class 2} \end{cases}$$

We can equivalently think of the two classes as "red" or "blue", as shown in Figure 2.1, with "red" as class 1 and "blue" as class 2.

We also need a way to determine whether an observation x is correctly classified by a weight vector w_k, where k denotes that we are currently in the k^{th} iteration of the algorithm. To this end, we define correct classification as follows:

Definition 4 (Correct Classification)**.** We say that a vector x with class y is correctly classified by a weight vector w_k if

$$y\langle w_k, x\rangle > 0.$$

We can think of this as a comparison of the signs of the true class, y, and predicted value $\hat{y} = \langle w_k, x\rangle$. Suppose that $y = 1$, if $\hat{y} > 0$ then the predicted class matches the true class so x is correctly classified by w_k. Similarly, if $y = -1$ and $\hat{y} < 0$, then we have $y\langle w_k, x\rangle > 0$. In cases of misclassification, we have that the signs of y and $\hat{y}$ are different. Suppose that $y = 1$ but the current weight vector predicts x to be in class 2. This means that $\hat{y} < 0$ and so $y \cdot \hat{y} < 0$. The same principle holds in the case when $y = -1$ but $\hat{y} > 0$, so $y \cdot \hat{y} < 0$. For completeness, we will use the convention that if $\langle w_k, x\rangle = 0$ (the point is on the decision boundary), then x is misclassified.

The crux to the Perceptron Algorithm is its *update rule*, which is simply the way we update our weight vector.

Definition 5 (Perceptron Update Rule)**.** If a vector x with class y is misclassified by a weight vector w_k, then the Perceptron Algorithm updates the weight vector according to the following rule:

$$w_{k+1} = w_k + y \cdot x.$$

At each iteration, the Perceptron searches for a misclassified point and then applies the update rule using that point and its corresponding y. While iterating through the data, if we encounter a correctly classified point we do not update w_k and continue the search for a misclassified point. Upon finding a misclassified vector, we then apply the update to w_k. If $y = 1$ we add on x to w_k, but if $y = -1$ we subtract x from w_k.

We also introduce the notion of the *bias term*, b. In some cases a dataset may not be linearly separable when our decision boundary must pass through the origin, but would be if we were able to apply some shift to it. The role of the bias term is to shift the intercept up or down after each update to w_k. The update rule to include the bias term is simply the update rule 5 with an additional update $b_{k+1} = b_k + y$, where b_k denotes the bias in the k^{th} iteration of the algorithm, and y is the class of the misclassified vector.

In the context of AI, this bias term may more commonly referred to as the threshold. For simplicity and clarity we will deal with the case where the data is roughly centred around the origin so will exclude the bias term from here on. In Figure 2.2 we show an example of the update rule without bias:

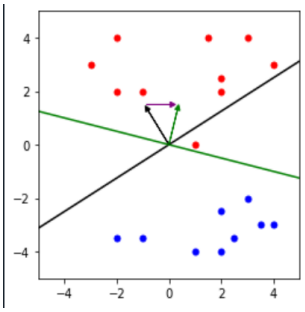

Figure 2.2: The black arrow is our current weight vector, and we update to the green arrow. The corresponding change in our weight vector is the purple arrow. The point $(1, 0)$ starts on the wrong side of the boundary, but is correctly classified after the update. In this case, all of the data is now correctly classified.

More explicitly, the calculation occurring in Figure 2.2 is:

$$w_{k+1} = w_k + y \cdot x = \begin{pmatrix} -0.75 \\ 1.5 \end{pmatrix} + 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 1.5 \end{pmatrix}.$$

2.3 Algorithm and Worked Example

We are now able to give the algorithm in pseudocode and an example of the full procedure in Figure 2.3:

Algorithm 1 Perceptron Algorithm

1: **Algorithm** PERCEPTRON-ALGORITHM(X)
2: Initialise $w = \underline{\mathbf{0}}$
3: **while** there are misclassified points **do**
4: **for** $i = 1, \ldots, X.length$ **do**
5: **if** $y_i \langle w, x_i \rangle \leq 0$ **then**
6: $w \leftarrow w + y_i \cdot x_i$

We give a line-by-line explanation of the pseudocode so we can see where all of the parts we have discussed so far are used. In Line 2 we initialise our weight vector as the zero vector $w_0 = \underline{\mathbf{0}}$. Line 3 is perhaps the most important line in the algorithm and we call it our stopping criterion. That is, a criterion that we aim to satisfy. For the Perceptron Algorithm, this will occur when our decision boundary separates our two classes of data i.e Definition 4 is correct for all observations in our dataset. Line 4 iterates through the data and then Line 5 checks whether the observation is correctly classified. In cases where all of the data is not correctly classified, Line 5 will fail and Line 6 updates the weight vector accordingly with the first misclassified point found using the update rule 5.

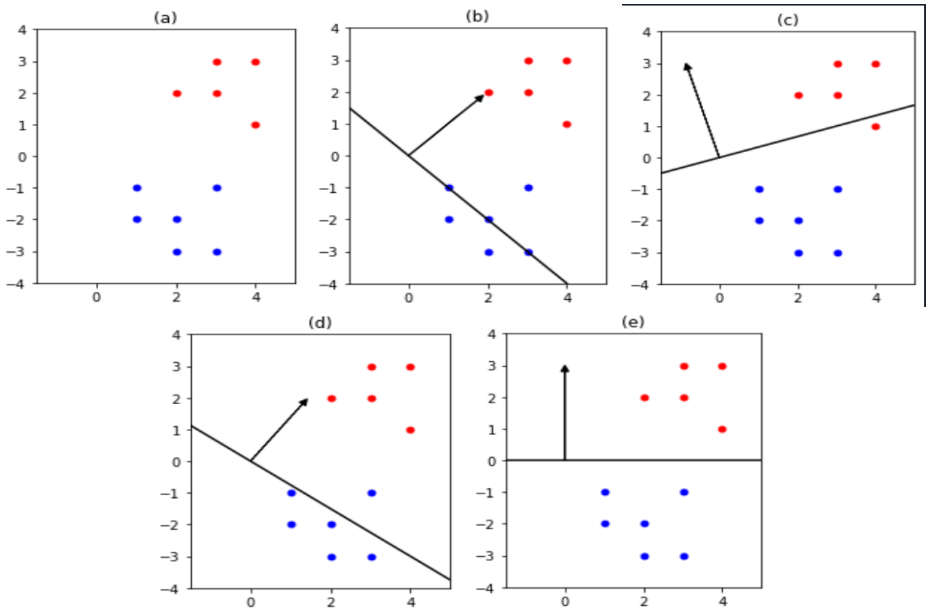

Figure 2.3: (a) We visualise our classified data. (b) We perform the first update to the weight vector, but $(3, -1)$ is misclassified. (c) We update w but $(4, 1)$ is now misclassified. (d) We update w but $(3, -1)$ is misclassified once again. (e) We update w and now all of the data is correctly classified.

2.4 Proofs

A natural question to ask at this point is: *"but why does this update rule work?"*. We now move onto explaining this in the context of its geometric interpretation using the inner product to understand why we use such a rule.

In the 2-dimensional setting, we can relate the normal of the boundary, w, to the vector of interest, x, with some angle $0 \leq \alpha \leq \pi$. Definition 2 tells us that the inner product of w with x is proportional to the cosine of the angle between them, so we have that $\cos\alpha \propto \langle w, x\rangle$. Figure 2.4 shows us how the value of α tells us whether a point is correctly classified or not.

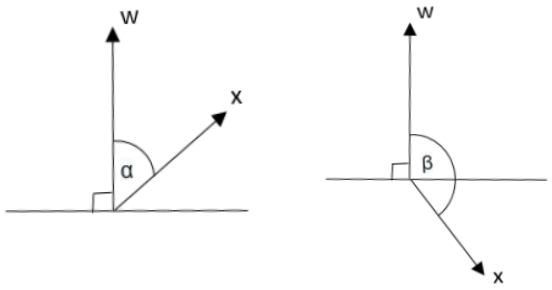

Figure 2.4: The horizontal line is a decision boundary, and x is the vector of interest. Suppose that the true class of x is $y = 1$. Left: x is correctly classified and we see that $\alpha < \frac{\pi}{2}$. Right: x is on the wrong side of the boundary and hence misclassified with $\frac{\pi}{2} < \beta$.

Theorem 6. *Let a vector x have corresponding class $y = 1$. Then, x is correctly classified if the angle α between w_k and x is between 0 and $\frac{\pi}{2}$.*

Proof. Suppose that $y = 1$, since we want y and $\hat{y}$ to be of the same sign, we want

$$\hat{y} = \langle w_k, x\rangle > 0.$$

From Definition 2, we know that $\cos\alpha \propto \langle w_k, x\rangle$, so

$$\hat{y} = \langle w_k, x\rangle > 0, \text{ which is equivalent to } \cos\alpha > 0.$$

Finally, since $\cos\alpha > 0$ and the inner product returns the smallest angle, $0 \leq \alpha < \frac{\pi}{2}$ and this concludes the proof. □

The proof for showing the case when $y = -1$ is near-identical. However, we now have $\langle w_k, x\rangle < 0$, so want $\cos\beta < 0$ as well. We can then conclude that $\beta > \frac{\pi}{2}$.

With this groundwork now in place, we can show in two ways why this update rule works. Our first proof uses inspiration from Definition 4, whereby we aim to show that the update increases the value of $y\langle w_k, x\rangle$. By increasing this quantity, we aim to make it positive - at which point x is correctly classified.

Theorem 7. *Let x be any vector misclassified by w_k. Then, the Perceptron update rule increases the value of $y\langle w_k x\rangle$.*

Proof.

$$\begin{aligned} y\langle w_{k+1}, x\rangle &= y\langle w_k + yx, x\rangle \text{ by update rule 5.} \\ &= y\langle w_k, x\rangle + y^2\langle x, x\rangle \\ &= y\langle w_k, x\rangle + \|x\|^2, \text{ because } y^2 = 1. \\ &\geq y\langle w_k, x\rangle, \text{ since } \|x\|^2 \geq 0. \end{aligned}$$

This shows that the update has increased the value of $y\langle w_k, x\rangle$. Hence, x is *closer* to being correctly classfied after this update and so concludes the proof. □

Our second proof is a new alternative way motivated by Definition 2 to show how the angle changes after the update. We motivate this proof by considering the change to $\langle w_{k+1}, x\rangle$:

$$\begin{aligned} \langle w_{k+1}, x\rangle &= \langle w_k + yx, x\rangle \text{ by update rule 5.} \\ &= \langle w_k, x\rangle + y\langle x, x\rangle \\ &= \langle w_k, x\rangle + y\|x\|^2 \end{aligned}$$

We see that determining whether $\langle w_{k+1}, x\rangle > \langle w_k, x\rangle$ or $\langle w_{k+1}, x\rangle < \langle w_k, x\rangle$ depends on the value of y. Remembering from Definition 2 that $\cos\alpha \propto \langle w, x\rangle$, this suggests that we can explore the update rule further by considering how it affects the angle between the w and x.

Let us first define a bit more notation. We let α be the original angle between the vectors in question. That is, between the weight vector w_k and the misclassified vector x. After the update, we let β be the new angle between w_{k+1} and x.

In the case where $y = 1$, we want to show that the update decreases the angle between w and x : $\beta \leq \alpha$. However, when $y = -1$, we show that the update increases the angle: $\alpha \leq \beta$. Since Definition 2 returns an angle α such that $0 \leq \alpha \leq \pi$, showing that $\beta \leq \alpha$ is equivalent to showing that $\cos\alpha \leq \cos\beta$ because cosine is a non-increasing function for $0 \leq \alpha \leq \pi$. Similarly, showing that $\alpha \leq \beta$, is equivalent to $\cos\beta \leq \cos\alpha$.

Theorem 8. *When a misclassified vector x has corresponding class $y = 1$, the Perceptron update rule decreases the angle between w_k and x. When $y = -1$, the update increases the angle between w_k and x.*

Proof. From Definition 2, we have that $\langle w_k, x\rangle = \|w_k\| \cdot \|x\| \cdot \cos\alpha$. Similarly, after the update we have that $\langle w_{k+1}, x\rangle = \|w_{k+1}\| \cdot \|x\| \cdot \cos\beta$.
Using update rule 5, we have that $w_{k+1} = w_k + yx$ so,

$$\cos\beta = \frac{\langle w_{k+1}, x\rangle}{\|w_{k+1}\| \cdot \|x\|} = \frac{\langle w_k + yx, x\rangle}{\|w_k + yx\| \cdot \|x\|}.$$

We now split into our two cases: $y = \pm 1$. First suppose that $y = 1$:

$$\frac{\langle w_k + yx, x\rangle}{\|w_k + yx\| \cdot \|x\|} = \frac{\langle w_k + x, x\rangle}{\|w_k + x\| \cdot \|x\|} = \frac{\langle w_k, x\rangle + \langle x, x\rangle}{\|w_k + x\| \cdot \|x\|} = \frac{\langle w_k, x\rangle + \|x\|^2}{\|w_k + x\| \cdot \|x\|}.$$

We aim to show that when $y = 1$, $\cos\alpha \leq \cos\beta$. By using the above, it is equivalent to showing that

$$\cos\alpha = \frac{\langle w_k, x\rangle}{\|w_k\| \cdot \|x\|} \leq \frac{\langle w_k, x\rangle + \|x\|^2}{\|w_k + x\| \cdot \|x\|} = \cos\beta.$$

Using the Triangle Inequality for Norms: $\|w_k + x\| \leq \|w_k\| + \|x\|$,

$$\cos\beta = \frac{\langle w_k, x\rangle + \|x\|^2}{\|w_k + x\| \cdot \|x\|} \geq \frac{\langle w_k, x\rangle + \|x\|^2}{\|w_k\| \cdot \|x\| + \|x\|^2}.$$

We now aim to show that

$$\cos\beta \geq \frac{\langle w_k, x\rangle + \|x\|^2}{\|w_k\| \cdot \|x\| + \|x\|^2} \geq \frac{\langle w_k, x\rangle}{\|w_k\| \cdot \|x\|} = \cos\alpha.$$

Multiplying both sides by $\|w_k\| \cdot \|x\| + \|x\|^2$,

$$\langle w_k, x\rangle + \|x\|^2 \geq \frac{\langle w_k, x\rangle \cdot \|w_k\| \cdot \|x\| + \langle w_k, x\rangle \cdot \|x\|^2}{\|w_k\| \cdot \|x\|},$$

then dividing the right side through by $\|w_k\| \cdot \|x\|$,

$$\langle w_k, x\rangle + \|x\|^2 \geq \langle w_k, x\rangle + \frac{\langle w_k, x\rangle \cdot \|x\|^2}{\|w_k\| \cdot \|x\|},$$

and subtracting $\langle w_k, x\rangle$ from both sides,

$$\|x\|^2 \geq \frac{\langle w_k, x\rangle \cdot \|x\|^2}{\|w_k\| \cdot \|x\|},$$

and finally dividing by $\|x\|^2$,

$$1 \geq \frac{\langle w_k, x\rangle}{\|w_k\| \cdot \|x\|} = \cos\alpha.$$

And this final line is true directly from the definition of cosine, so we have shown that when $y = 1$, $\beta \leq \alpha$. We now move onto the case when $y = -1$. The proof is more or less the same, but we must be careful when using the Triangle Inequality. Now suppose that $y = -1$:

$$\cos\beta = \frac{\langle w_k + yx, x\rangle}{\|w_k + yx\| \cdot \|x\|} = \frac{\langle w_k - x, x\rangle}{\|w_k - x\| \cdot \|x\|} = \frac{\langle w_k, x\rangle - \langle x, x\rangle}{\|w_k - x\| \cdot \|x\|} = \frac{\langle w_k, x\rangle - \|x\|^2}{\|w_k - x\| \cdot \|x\|}.$$

We aim to show that when $y = -1$, $\cos\beta \leq \cos\alpha$. This is equivalent to showing that

$$\cos\alpha = \frac{\langle w_k, x\rangle}{\|w_k\| \cdot \|x\|} \leq \frac{\langle w_k, x\rangle - \|x\|^2}{\|w_k - x\| \cdot \|x\|} = \cos\beta.$$

Again using the Triangle Inequality for Norms, but note that when $y = -1$ we have that $\|w_k - x\| \geq \|w_k\| - \|x\|$. Applying this yields

$$\cos\beta = \frac{\langle w_k, x\rangle - \|x\|^2}{\|w_k - x\| \cdot \|x\|} \leq \frac{\langle w_k, x\rangle - \|x\|^2}{\|w_k\| \cdot \|x\| - \|x\|^2}.$$

We now aim to show that

$$\cos\beta \leq \frac{\langle w_k, x\rangle - \|x\|^2}{\|w_k\| \cdot \|x\| - \|x\|^2} \leq \frac{\langle w_k, x\rangle}{\|w_k\| \cdot \|x\|} = \cos\alpha.$$

Multiplying both sides by $\|w_k\| \cdot \|x\| - \|x\|^2$,

$$\langle w_k, x\rangle - \|x\|^2 \leq \frac{\langle w_k, x\rangle \cdot \|w_k\| \cdot \|x\| - \langle w_k, x\rangle \cdot \|x\|^2}{\|w_k\| \cdot \|x\|},$$

then dividing the right side through by $\|w_k\| \cdot \|x\|$,

$$\langle w_k, x\rangle - \|x\|^2 \leq \langle w_k, x\rangle - \frac{\langle w_k, x\rangle \cdot \|x\|^2}{\|w_k\| \cdot \|x\|},$$

and subtracting $\langle w_k, x\rangle$ from both sides,

$$-\|x\|^2 \leq -\frac{\langle w_k, x\rangle \cdot \|x\|^2}{\|w_k\| \cdot \|x\|},$$

and finally dividing by $-\|x\|^2$,

$$1 \geq \frac{\langle w_k, x\rangle}{\|w_k\| \cdot \|x\|} = \cos\alpha.$$

As before, this last line is true directly from the definition of cosine. This has shown that the angle has increased, and so concludes the proof. □

2.5 Mistake Bound

The final proof of this chapter will be crucial for the rest of this dissertation as it lays the framework which we can adapt to various other scenarios.

Theorem 9 (Novikoff (1963))**.** *The number of updates the Perceptron makes is upper bounded by* $(\frac{R}{\gamma})^2$.

Proof. Let x_k be a misclassified vector in the k^{th} round, with class y_k. Furthermore, let θ^* be normal to a hyperplane that separates our data, with $\|\theta^*\|^2 = 1$. We now aim to find a lower bound for $\|w_{k+1}\|$.

$$\begin{aligned}
\langle w_{k+1}, \theta^*\rangle &= \langle w_k + y_k \cdot x_k, \theta^*\rangle \text{ using update rule 5.} \\
&= \langle w_k, \theta^*\rangle + y_k\langle x_k, \theta^*\rangle \\
&\geq \langle w_k, \theta^*\rangle + \gamma \text{ by the assumption that } \gamma \leq y_k\langle \theta^*, x_k\rangle.
\end{aligned}$$

By induction on k and since $w_0 = \underline{\mathbf{0}}$,

$$\langle w_{k+1}, \theta^* \rangle \geq k\gamma.$$

Applying the Cauchy-Schwarz inequality $\|w_{k+1}\| \cdot \|\theta^*\| \geq \langle w_{k+1}, \theta^* \rangle$,

$$\|w_{k+1}\| \cdot \|\theta^*\| \geq k\gamma,$$

and since $\|\theta^*\| = 1$, we now have our bound

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

Second Part: we find an upper bound for $\|w_{k+1}\|$.

$$\begin{aligned}
\|w_{k+1}\|^2 &= \|w_k + y_k x_k\|^2 \\
&= \|w_k\|^2 + 2y_k\langle w_k, x_k\rangle + y_k^2\|x_k\|^2 \\
&\leq \|w_k\|^2 + y_k^2\|x_k\|^2 = \|w_k\|^2 + \|x_k\|^2 \text{ , because } y_k\langle w_k, x_k\rangle \leq 0. \\
&\leq \|w_k\|^2 + R^2 \text{ , since } \|x_k\| \leq R.
\end{aligned}$$

Using induction on w_k,

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

then square rooting both sides,

$$\|w_{k+1}\| \leq \sqrt{k}R,$$

and combining these two results,

$$k\gamma \leq \|w_{k+1}\| \leq \sqrt{k}R.$$

Comparing the left and right sides,

$$k\gamma \leq \sqrt{k}R,$$

then finally re-arranging to

$$k \leq (\frac{R}{\gamma})^2,$$

concluding the proof. □

This concludes our analysis of the Perceptron Algorithm. In the subsequent chapters, we will explore modifications and extensions of the Perceptron. In reality, the Perceptron is a very stylised classification algorithm and is not wholly satisfying, but with some of these modifications we can create something that is more useful.

Chapter 3

The Greedy Perceptron Algorithm

3.1 Motivation for the Greedy Perceptron

In this chapter we introduce a modification of the Perceptron Algorithm using the *greedy algorithm* framework. By this, we mean that the algorithm chooses the best option at each iteration without consideration for the globally optimal solution.

A question the reader may have asked themselves when looking at update rule 5 is: "Why do we add or subtract *one* unit of the misclassified vector to w?". In truth, this is a relatively arbitrary decision but has been useful for the sake of simplicity. Therefore, we could re-construct update rule 5 to something like this:

Definition 10 (Update Rule with Learning Rate)**.** Let $\eta > 0$ be some constant. When the Perceptron finds a misclassified point x with corresponding class y, the update to the weight vector w_k is:

$$w_{k+1} = w_k + \eta \cdot y \cdot x.$$

The difference between the old update rule and this new one is that previously we had that $\eta = 1$, but now we can choose any $\eta > 0$. We will call this η the *learning rate*, and visualise these different learning rates in Figure 3.1. The only change we must make to include this fixed learning rate is using update rule 10 instead in line 6 of Algorithm 1.

As with any tuning parameter, we have a trade-off. In the case where we choose a large η, we run the risk of making such a large change to w that some other point becomes misclassified after the update. On the other hand, when η is too small, the misclassified point may remain misclassified even after the update so we need to take more steps for that point to be classified correctly. We move onto giving examples where we see that different learning rates reach solutions in a different number of iterations.

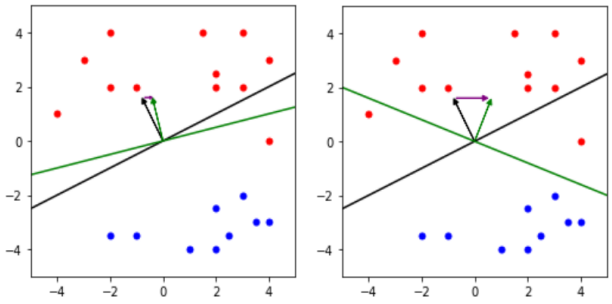

Figure 3.1: The black line is the initial boundary, and the green line is the boundary after the update. Left: An example where η is too small so even after the update the point $(4, 0)$ is still misclassified. Right: An example where η is too large, so the point $(4, 0)$ is correctly classified but $(-4, 1)$ is now misclassified.

3.2 Loss Functions

We now introduce some more theory: the *Loss Function*. In the context of classification problems, loss functions are a useful way to measure how well a decision boundary performs in classifying each of the points. The most simple of them all is the *0-1 Loss Function* defined as:

Definition 11 (0-1 Loss Function)**.** Let $\mathbb{1}$ denote an indicator function that takes value 1 if the predicted value $\hat{y}_i$ matches the true class y_i, and 0 otherwise. Then the loss of w is

$$\mathcal{L}(w) = \sum_{i=1}^{n} \mathbb{1}[\hat{y}_i \neq y_i].$$

Hence, we can think of the 0-1 Loss Function as the number of misclassified points. Suppose that $\mathcal{L}(w) = 10$, then we say that w "has loss 10".

In Figure 3.2 we give two examples which use different datasets to demonstrate the rates of convergence for various learning rates. For no particular reason, we choose learning rates $\eta = 0.5, 1, 2, 4$.

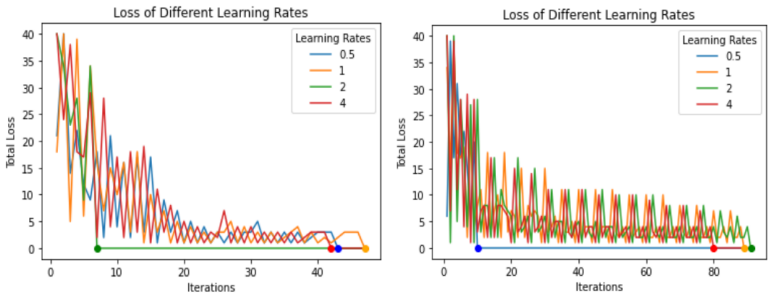

Figure 3.2: The coloured dots represent the termination of the algorithm. Left: An example where $\eta = 2$ reaches a solution the fastest. Right: On a different dataset we find that $\eta = 0.5$ reaches a solution the fastest.

As we can see, the different learning rates converge to a solution in a different number of iterations. By noticing that the loss sometimes increases after an update, we see that there are some updates that put us in a worse position than we started from. Using this as inspiration, we introduce a Greedy Perceptron Algorithm that works as follows: we select a finite set S of l different learning rates, $|S| = l$, and update w_k by choosing the learning rate that minimises loss after the update. In Figure 3.3 we visualise what this might look like.

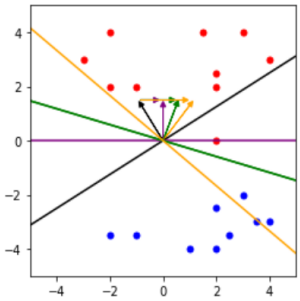

Figure 3.3: The point $(2, 0)$ is misclassified. Starting from the black normal, the Greedy Perceptron then chooses between the purple, green, or orange normals as the update. Since the green boundary has the smallest loss, we pick it as our update. In this case, the boundary now correctly classifies all of the data.

3.3 The Algorithm

With the framework in place, we can now give the algorithm.

Algorithm 2 Greedy Perceptron Algorithm

1: **Algorithm** GREEDY-PERCEPTRON-ALGORITHM(X, S)
2: Initialise $w_0 = \underline{\mathbf{0}}$
3: **while** there are misclassified points **do**
4: **for** $i = 1, \ldots, X.length$ **do**
5: **if** $y_i \langle w, x_i \rangle \leq 0$ **then**
6: Choose greedy learning rate $\eta^* \in S$
7: $w \leftarrow w + \eta^* \cdot y_i \cdot x_i$

This looks incredibly similar to the standard Perceptron given in Chapter 2, with the only difference being the additional choice of learning rates. More formally, in line 6 we choose η^* according to the following rule:

Definition 12 (Greedy Learning rate)**.** Let S be the set of learning rates chosen for the Greedy Perceptron. When a misclassified point x with corresponding class y is found, the Greedy Perceptron calculates the greedy learning rate by

$$\eta^* = \underset{\eta \in S}{\operatorname{argmin}} \mathcal{L}(w_k + \eta \cdot y \cdot x).$$

Definition 13 (Greedy Update Rule)**.** Let x be a misclassified vector with corresponding class y. After calculating the greedy learning rate η^*, the Greedy Perceptron updates the weight vector according to the following rule:

$$w_{k+1} = w_k + \eta^* \cdot y \cdot x.$$

So how does this new Greedy Perceptron compare to the Perceptron with different fixed learning rates? In Figure 3.4 we give an example comparing the Greedy Perceptron to the standard Perceptron with different learning rates.

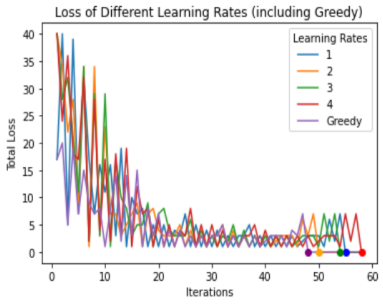

Figure 3.4: An example where the Greedy Perceptron terminates in 48 iterations, which is faster than any of the Perceptrons with fixed learning rate.

We now move onto showing an adaptation of Theorem 9 in context of this Greedy Perceptron to derive the mistake bound. Before that, however, we should see whether the result would change with a *fixed* learning rate, as in to Definition 10.

Theorem 14. *The upper bound of mistakes for the Perceptron is independent of the learning rate.*

Proof. We provide a near-identical proof to what would be required to prove the result from Theorem 9, but take care when applying the update rule. We let θ^* be the normal to a decision boundary that linearly separates our data with $\|\theta^*\|^2 = 1$. We now aim to find a lower bound for $\|w_{k+1}\|$. Let x_k be a misclassified vector in the k^{th} round of the algorithm with class y_k.

$$\begin{aligned}\langle w_{k+1}, \theta^* \rangle &= \langle w_k + \eta \cdot y_k \cdot x_k, \theta^* \rangle \text{ using update rule 10.} \\ &= \langle w_k, \theta^* \rangle + \eta y_k \langle x_k, \theta^* \rangle \\ &\geq \langle w_k, \theta^* \rangle + \eta\gamma \text{ by the assumption that } \gamma \leq y_k \langle \theta^*, x_k \rangle.\end{aligned}$$

By induction on k and since $w_0 = \underline{\mathbf{0}}$,

$$\langle w_{k+1}, \theta^* \rangle \geq \eta k \gamma.$$

Applying the Cauchy-Schwarz inequality $\|w_{k+1}\| \cdot \|\theta^*\| \geq \langle w_{k+1}, \theta^* \rangle$,

$$\|w_{k+1}\| \cdot \|\theta^*\| \geq \eta k \gamma,$$

and since $\|\theta^*\| = 1$, we now have our bound

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

Second Part: we find an upper bound for $\|w_{k+1}\|$:

$$\begin{aligned}
\|w_{k+1}\|^2 &= \|w_k + \eta y_k x_k\|^2 \\
&= \|w_k\|^2 + 2\eta y\langle w_k, x_k\rangle + \eta^2 {y_k}^2 \|x_k\|^2 \\
&\leq \|w_k\|^2 + \eta^2 {y_k}^2 \|x_k\|^2 = \|w_k\|^2 + \eta^2 \|x_k\|^2 \text{ , because } y_k\langle w_k, x_k\rangle \leq 0. \\
&\leq \|w_k\|^2 + \eta^2 R^2 \text{ , since } \|x_k\| \leq R.
\end{aligned}$$

Using induction on w_k,

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

then square rooting both sides,

$$\|w_{k+1}\| \leq \eta\sqrt{k}R,$$

and combining these two results,

$$\eta k\gamma \leq \|w_{k+1}\| \leq \eta\sqrt{k}R.$$

Comparing the left and right sides,

$$\eta k\gamma \leq \eta\sqrt{k}R,$$

then dividing both sides by η and re-arranging,

$$k \leq (\frac{R}{\gamma})^2.$$

This shows the independence of the bound on η, concluding the proof. □

3.4 Mistake Bounds for the Greedy Perceptron

We now move onto showing how the result of Theorem 9 changes when we use the *variable* learning rate method for the Greedy Perceptron. Suppose that the Greedy Perceptron takes k rounds to learn a separating hyperplane. Let η_j denote the learning rate used by the Greedy Perceptron in the j^{th} round.

Theorem 15. *The learning rates used by the Greedy Perceptron $\eta_1, \eta_2, \ldots, \eta_k$, satisfy*

$$\frac{(\sum_{i=1}^k \eta_i)^2}{\sum_{i=1}^k \eta_i^2} \leq (\frac{R}{\gamma})^2.$$

Proof. Similarly to before, we first find a lower bound for $\|w_{k+1}\|$. Let x_k be a misclassified vector in the k^{th} round with class y_k.

$$\begin{aligned}\langle w_{k+1}, \theta^* \rangle &= \langle w_k + \eta_k \cdot y_k \cdot x_k, \theta^* \rangle \\ &\geq \langle w_k, \theta^* \rangle + \eta_k \gamma \\ &\geq \langle w_{k-1}, \theta^* \rangle + \eta_k \gamma + \eta_{k-1}\gamma \text{ by repeating the previous steps on } w_k. \\ &\geq \langle w_{k-2}, \theta^* \rangle + \gamma(\eta_k + \eta_{k-1} + \eta_{k-2})\end{aligned}$$

By induction on k and since $w_0 = \underline{\mathbf{0}}$,

$$\langle w_{k+1}, \theta^* \rangle \geq \gamma \sum_{i=1}^{k} \eta_i.$$

Using the Cauchy-Schwarz inequality as before,

$$\|w_{k+1}\| \geq \gamma \sum_{i=1}^{k} \eta_i.$$

For the second part, we once again find an upper bound for $\|w_{k+1}\|$:

$$\begin{aligned}\|w_{k+1}\|^2 &= \|w_k + \eta_k y_k x_k\|^2 \\ &= \|w_k\|^2 + 2\eta_k y_k \langle w_k, x_k \rangle + \eta_k^2 y_k^2 \|x_k\|^2 \\ &= \|w_{k-1} + \eta_{k-1} y_{k-1} x_{k-1}\|^2 + 2\eta_k y_k \langle w_k, x_k \rangle + \eta_k^2 y_k^2 \|x_k\|^2 \\ &= \|w_{k-1}\|^2 + 2\eta_{k-1} y_{k-1} \langle w_{k-1}, x_{k-1} \rangle + \eta_{k-1}^2 y_{k-1}^2 \|x_{k-1}\|^2 + 2\eta_k y_k \langle w_k, x_k \rangle + \eta_k^2 y_k^2 \|x_k\|^2 \\ &= 2\sum_{i=1}^{k} \eta_i y_i \langle w_i, x_i \rangle + \sum_{i=1}^{k} \eta_i^2 y_i^2 \|x_i\|^2 \text{ , using induction on } w \text{ again.} \\ &\leq \sum_{i=1}^{k} \eta_i^2 \|x_i\|^2 \text{ , since } y_i \langle w_i, x_i \rangle \leq 0 \text{ because each } x_i \text{ is misclassified.} \\ &\leq \sum_{i=1}^{k} \eta_i^2 R^2 \text{ , using the definition of the radius } R.\end{aligned}$$

From which we have that

$$\|w_{k+1}\| \leq \sqrt{\sum_{i=1}^{k} \eta_i^2 R^2},$$

and we now combine these bounds together again to have that

$$\gamma \sum_{i=1}^{k} \eta_i \leq \|w_{k+1}\| \leq \sqrt{\sum_{i=1}^{k} \eta_i^2 R^2}.$$

Squaring both sides we obtain

$$\gamma^2 (\sum_{i=1}^{k} \eta_i)^2 \leq R^2 \sum_{i=1}^{k} \eta_i^2,$$

and finally re-arrange to

$$\frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} \leq (\frac{R}{\gamma})^2.$$

□

We note that for a fixed η, this result degenerates to the result from Theorem 9. Suppose that we have a fixed $\eta_i = c$, then

$$\frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} = \frac{(\sum_{i=1}^{k} c)^2}{\sum_{i=1}^{k} c^2} = \frac{(kc)^2}{kc^2} = \frac{k^2c^2}{kc^2} = k \leq (\frac{R}{\gamma})^2.$$

Theorem 16. *The learning rates used by the Greedy Perceptron $\eta_1, \eta_2, \ldots, \eta_k$, satisfy*

$$k \cdot (\frac{\eta_{min}}{\eta_{max}})^2 \leq \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2},$$

where η_{max} and η_{min} denote the maximum and minimum of set S, respectively.

Proof.

$$\frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} \geq \frac{(\sum_{i=1}^{k} \eta_{\min})^2}{\sum_{i=1}^{k} \eta_i^2} \geq \frac{(\sum_{i=1}^{k} \eta_{\min})^2}{\sum_{i=1}^{k} \eta_{\max}^2}.$$

Simplifying this expression,

$$\frac{(\sum_{i=1}^{k} \eta_{\min})^2}{\sum_{i=1}^{k} \eta_{\max}^2} = \frac{(k \cdot \eta_{\min})^2}{k \cdot \eta_{\max}^2} = k \cdot (\frac{\eta_{\min}}{\eta_{\max}})^2,$$

so we conclude that we have a lower bound of

$$k \cdot (\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2}.$$ □

Corollary 17. *The learning rates used by the Greedy Perceptron $\eta_1, \eta_2, \ldots, \eta_k$, satisfy*

$$(\frac{\eta_{min}}{\eta_{max}})^2 \leq \frac{1}{k} \cdot (\frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} = \frac{(Average\ of\ \eta_i)^2}{Average\ of\ \eta_i^2}.$$

Proof. From Theorem 16 we know that

$$k \cdot (\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2},$$

which is equivalent to

$$(\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{1}{k} \cdot \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2}.$$

Distributing the $\frac{1}{k}$ term yields

$$(\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{(\frac{1}{k})^2}{(\frac{1}{k})} \cdot \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2},$$

and now combining these fractions together,

$$(\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{(\frac{1}{k} \sum_{i=1}^{k} \eta_i)^2}{\frac{1}{k} \sum_{i=1}^{k} \eta_i^2} = \frac{(\text{Average of } \eta_i)^2}{\text{Average of } \eta_i^2}.$$ □

With these bounds, we can now derive a more important result: the mistake bound for the Greedy Perceptron.

Theorem 18. *The Greedy Perceptron has an upper bound on the number of mistakes it makes of*

$$k \leq (\frac{\eta_{max}}{\eta_{min}})^2 \cdot (\frac{R}{\gamma})^2.$$

Proof. Theorems 15 and 16 tell us that

$$k \cdot (\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} \text{ and } \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} \leq (\frac{R}{\gamma})^2,$$

so combining these bounds we obtain

$$k \cdot (\frac{\eta_{\min}}{\eta_{\max}})^2 \leq \frac{(\sum_{i=1}^{k} \eta_i)^2}{\sum_{i=1}^{k} \eta_i^2} \leq (\frac{R}{\gamma})^2.$$

Comparing the left and right sides,

$$k \cdot (\frac{\eta_{\min}}{\eta_{\max}})^2 \leq (\frac{R}{\gamma})^2,$$

which we re-arrange to

$$k \leq (\frac{\eta_{\max}}{\eta_{\min}})^2 \cdot (\frac{R}{\gamma})^2,$$

and this completes the proof. □

This tells us that the Greedy Perceptron has a trade-off of reducing the number of actual updates it makes by having more learning rates to choose from, but at the cost of increasing the upper bound. We also notice that this bound coincides with the standard Perceptron mistake bound, Theorem 9, when $\eta_{\min} = \eta_{\max}$, which can only be true when we only have one option for each η_i.

One question this does raise is that this bound only depends on the maximum and minimum of set S, so why could we not just use lots of η between the minimum learning rate, η_{min}, and the maximum, η_{max}? However, by using many learning rates, it will dramatically increase the computational cost of the algorithm. With more and more learning rates, we approach a scenario where we are almost performing *Exact Line Search*. While it may lead to fewer iterations to find a solution, the overall computational cost will still be much higher.

As we can see, in some cases the Greedy Perceptron finds a solution faster than the standard Perceptron. This is not to say that this modification will guarantee that we find a solution faster every time, however, the option to choose from various updates can prevent putting us in a much worse position when restricted to a single learning rate. The pitfall with greedy algorithms lies in the fact that always choosing the locally-optimal decision may not lead to us reaching a solution faster. Sometimes it may be better to choose an η that puts us in a slightly worse position than we started from, but then allows us to reach a solution in fewer steps from there. Therefore, we should not think of this Greedy Perceptron as a silver bullet to all Perceptron problems, but as seen, the flexibility with the learning rates can allow us to reach a solution faster.

Chapter 4

The Fuzzy Perceptron

4.1 Motivation for the Fuzzy Perceptron

In this chapter, we finally reconcile an issue with the variants of the Perceptron discussed so far. We introduce a variant called the *Fuzzy Perceptron* that will allow us to partially deal with some cases of linearly inseparable data. In particular, when the removal of a subset of the data now makes the remaining data linearly separable. We get the name *Fuzzy Perceptron* due to the fact that we will not update on what we will call *fuzzy vectors*. We will formalise what this means later, but for now we can think of fuzzy vectors as data points that lie roughly in the middle of the two classes of data. Previously, we did not consider the level of certainty with our prediction - only whether the vector was classified correctly or not. The general idea of the Fuzzy Perceptron is to assign probabilities that each vector belongs to Class 1 or Class 2 before we run the standard Perceptron. In a sense, first the Fuzzy Perceptron "ignores" the fuzzy data and then uses the standard Perceptron on the remaining non-fuzzy data. This now gives us the notion of a margin as before, but this time it is the margin to the non-fuzzy vectors. Since we can think of the Fuzzy Perceptron as the standard Perceptron with a subset of the data, it is worth noting that we are still able to guarantee convergence of the algorithm to find a solution. The upshot of this being that even in cases where the data is linearly inseparable, we are now still able to terminate the algorithm with an approximate solution for the *entire* dataset.

4.2 Framework

We use the general framework suggested in Keller and Hunt (1985), but will modify some of the notation used for consistency with the previous chapters. Firstly, we should denote the probabilities that a vector x belongs to Class 1 or Class 2 as $p_1(x)$ and $p_2(x)$, respectively, and we will call these probabilities *membership values*.

Definition 19 (Fuzzy Vector)**.** Given some small $\delta > 0$ and some vector x with membership values $p_1(x)$ and $p_2(x)$, x is fuzzy if

$$p_1(x), p_2(x) \in [0.5 - \delta, 0.5 + \delta].$$

Definition 20 (Fuzzy Update Rule)**.** Given the membership values $p_1(x)$ and $p_2(x)$, a learning rate $\eta > 0$, and some $m > 1$, when a misclassified point x with class y is found, the Fuzzy Perceptron updates the weight vector as follows:

$$w_{k+1} = w_k + \eta \cdot |p_1(x) - p_2(x)|^m \cdot y \cdot x.$$

We see that this looks very similar to update rule 10, however, we also scale our update by a factor of $|p_1(x) - p_2(x)|^m$. When one $p_1(x)$ is much larger than $p_2(x)$, or vice-versa, we perform larger updates. Since $p_1(x) + p_2(x) = 1$, $0 \leq |p_1(x) - p_2(x)| \leq 1$ so increasing m scales down the size of the update. We will use 6 ideas to motivate the reasoning behind the function that will allow us to calculate membership values:

1. It should be 1 if the vector is equal to the mean of its class.
2. It should be 0.5 if the vector is equal to the mean of the other class.
3. It should be near 0.5 if the vector is equidistant from the two means.
4. It should never be less than 0.5 or greater than 1.
5. As a vector gets closer to its mean and farther from the other mean, the membership value should approach 1 exponentially quickly.
6. It should depend on relative distances from the means of the classes rather than absolute distances.

Definition 21 (Fuzzy Membership Values)**.** Let x be some vector that belongs to class 1 and has distance d_1 and d_2 from the centres of mass of class 1 and 2, respectively. Further, let d denote the distance between d_1 and d_2 and $c > 0$ be some constant. Then, the probability that x belongs to each class is

$$p_1(x) = 0.5 + \frac{e^{\frac{c(d_2 - d_1)}{d}} - e^{-c}}{2(e^c - e^{-c})}, \text{ and } p_2(x) = 1 - p_1(x).$$

To calculate the membership values for a vector that belongs to Class 2, we would swap d_1 and d_2, and this would then calculate $p_2(x)$.

Algorithm 3 Fuzzy Perceptron Algorithm

```
1: Algorithm FUZZY-PERCEPTRON-ALGORITHM(X, η, m, δ)
2:     Initialise w = 0
3:     while there exist misclassified non-fuzzy vectors do
4:         for i = 1, ..., X.length do
5:             if y_i⟨w, x_i⟩ ≤ 0 and x_i is not fuzzy then
6:                 w ← w + η · |p_1(x_i) − p_2(x_i)|^m · y_i · x_i
```

4.3 Proofs

Theorem 22. (Fuzzy Perceptron Mistake Bound) *Suppose that the margin created by the choice of δ in the Fuzzy Perceptron is γ^*. Then, the Fuzzy Perceptron has an upper bound on the number of mistakes it makes of $(\frac{R}{\gamma^*})^2$.*

Proof. Since the Fuzzy Perceptron can be thought of as the standard Perceptron using a subset of the data, this result is simply an application of Theorem 9 using margin $\gamma = \gamma^*$. □

We now move onto showing a new geometric interpretation of the Fuzzy Perceptron. By construction, the Fuzzy Perceptron creates a margin γ^* that now allows us to separate the non-fuzzy vectors. We are able to quantify the minimum amount that the solutions can vary by considering the worst case, shown in Figure 4.1.

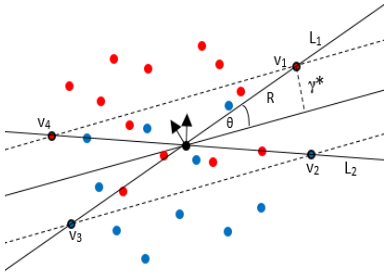

Figure 4.1: An example of the most restrictive case. The Fuzzy Perceptron "ignores" any data within γ^* of the boundary.

Here we see an example of the most restrictive case, where v_1, v_2, v_3, v_4 lie as-good-as on the margin created, with $\|v_1\| = \|v_2\| = \|v_3\| = \|v_4\| = R$. All fuzzy data is contained in the margin, and hyperplanes L_1 and L_2 separate the non-fuzzy data. This gives rise to the following theorem:

Theorem 23. (Fuzzy Perceptron Angle Approximation) *Suppose that the margin induced by the choice of δ is γ^*. Then, the decision boundary found by the Fuzzy Perceptron can vary by at least $2\sin^{-1}(\frac{\gamma^*}{R})$ radians.*

Proof. We can see directly from Figure 4.1 that the angle between hyperplanes L_1 and L_2 is $2\theta = 2\sin^{-1}(\frac{\gamma^*}{R})$. □

In summary, the Fuzzy Perceptron now allows us to find approximate solutions even when the data is linearly inseparable. In reality, most datasets won't be linearly separable, so the Fuzzy Perceptron is an extremely useful way to overcome one of the most fundamental flaws with the Perceptron algorithm. However, we must be careful in cases where the data has a large proportion of fuzzy vectors (γ^* only slightly smaller than R), as Theorem 23 tells us that a feasible solution may then be far from the "best" solution.

Chapter 5

The Margin-Fuzzy Perceptron

5.1 Motivation

In this chapter, we introduce a new variant of the Fuzzy Perceptron which we will call the *Margin-Fuzzy Perceptron*. As mentioned in Chapter 4, by the construction of the Fuzzy Perceptron we can think of it as a two step procedure: firstly, choose a δ that will ignore overlapping fuzzy data; then, run the standard Perceptron only using the non-fuzzy data. In this chapter, we posit a re-formulation of the Fuzzy Perceptron that allows us to more directly choose the size of the margin we use. Furthermore, rather than ignoring the same subset of the data at all iterations, the Margin-Fuzzy Perceptron ignores different subsets of data at each iteration.

We also take inspiration from the Soft-Margin SVM which terminates with a larger margin, but this comes at the cost of misclassifying some of the data. The analogue to the Margin-Fuzzy Perceptron is that a larger margin can contain the fuzzy data in more ways so the algorithm can terminate faster, but we will be using a smaller subset of the data so the final boundary will be less representative of the entire dataset. Figure 5.1 shows the effect of changing the margin size.

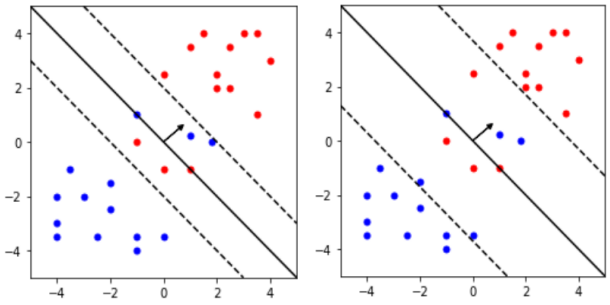

Figure 5.1: Left: An example of the smallest margin that contains all of the overlapping data. Right: The additional slack now includes more data points inside of the margin.

5.2 Notation

In this new formulation, we will define the fuzzy region we ignore to be the margin around the decision boundary. Recall that when $\|w\| = 1$, $\langle w, x \rangle$ is the *signed* perpendicular distance of x from the boundary. We now choose the margin ϵ directly, ensuring that $\epsilon > \gamma*$, where γ^* is the smallest possible margin that includes all of the overlapping data.

5.3 The Algorithm

Now that we have motivated the reasoning for the Margin-Fuzzy Perceptron, we can explicitly give the algorithm:

Algorithm 4 Margin-Fuzzy Perceptron Algorithm

1: **Algorithm** MARGIN-FUZZY-PERCEPTRON-ALGORITHM(X, ϵ, η)
2: Initialise $w = \underline{\mathbf{0}}$
3: **while** there exist misclassified vectors at least ϵ from the boundary **do**
4: **for** $i = 1, \ldots, X.length$ **do**
5: **if** $y_i \langle w_k, x_i \rangle \leq 0$ **then**
6: $w \leftarrow w + \eta \cdot y_i \cdot x_i$

We could equivalently write our stopping criteria as *"while there exists misclassified vectors x with $\langle w, x \rangle < -\epsilon$"* by combining Lines 3 and 5. With this in mind, Figure 5.3 shows what this stopping criterion looks like:

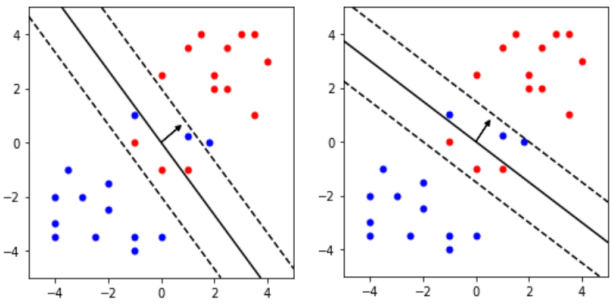

Figure 5.2: Left: An example where an overlapping point, (2,0), is misclassified and outside of the margin. Right: All overlapping points now lie within the margin so the algorithm terminates.

5.4 Proofs

With this new framework now in place, we can proceed to prove the claims made so far in this chapter.

Theorem 24. (Margin-Fuzzy Perceptron Mistake Bound) *By using a margin ϵ that is larger than the smallest required, γ^*, we lower the mistake bound of the Margin-Fuzzy Perceptron.*

Proof. The proof comes directly from comparing the results of Theorem 9 using margins γ^* and ϵ which gives us upper bounds of $(\frac{R}{\gamma^*})^2$ and $(\frac{R}{\epsilon})^2$. Then, since we have $\gamma^* \leq \epsilon$, $(\frac{R}{\epsilon})^2 \leq (\frac{R}{\gamma^*})^2$ and this completes the proof. □

For the Margin-Fuzzy Perceptron we are able to give a geometric interpretation similar to Theorem 23, as shown in Figure 5.3:

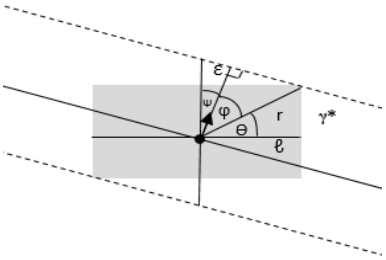

Figure 5.3: With a large enough margin, the Margin-Fuzzy Perceptron contains the shaded region of overlapping data.

Theorem 25. (Margin Fuzzy Angle Appoximation) *Suppose that we are able to cover all of the overlapping data with a blanket centred on the origin of width $2l$ and height $2\gamma^*$. Then, the Margin-Fuzzy Perceptron returns a solution that is able to vary by $2\psi = \pi - 2\cos^{-1}(\frac{l}{r}) - 2\cos^{-1}(\frac{\epsilon}{r})$ radians, where $r = \sqrt{l^2 + \gamma^{*2}}$.*

Proof. We can see from the Figure 5.3 that $\theta = \cos^{-1}(\frac{l}{r})$, $\phi = \cos^{-1}(\frac{\epsilon}{r})$, and so conclude that $\psi = \frac{\pi}{2} - \cos^{-1}(\frac{l}{r}) - \cos^{-1}(\frac{\epsilon}{r})$. By symmetry, the Margin-Fuzzy Perceptron can vary by 2ψ radians. □

We see that for the same margin, the Fuzzy Perceptron has a larger range of hyperplanes that separate the data than the Margin-Fuzzy Perceptron. Consider the case of the smallest margin $\epsilon = \gamma^*$. Then the Margin-Fuzzy Perceptron has no variation in possible separating hyperplanes, whereas Theorem 23 tells us that the solution to the Fuzzy Perceptron can vary by at least $2\sin^{-1}\frac{\gamma^*}{R}$. However, the benefit of the Margin-Fuzzy Perceptron comes from examining the Bias-Variance trade-off: while the Fuzzy Perceptron has a wider range of solutions, it also has a much higher variance when adding an additional point. In contrast, the while the Margin-Fuzzy Perceptron may take longer to find a solution, it will be less responsive to additional data points.

Bibliography

Cortes, C. and V. Vapnik (1995). Support-vector networks. *Machine Learning* 20(3), 273–297.

Keller, J. M. and D. J. Hunt (1985). Incorporating fuzzy membership functions into the perceptron algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (6), 693–699.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5, 115–133.

Novikoff, A. B. (1963). On convergence proofs for perceptrons. Technical report, Stanford Research Inst Menlo Park CA.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65(6), 386–408.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA* .

Appendix A

Python Code

A.1 Implementation of the Perceptron

```
import numpy as np

# Perceptron Algorithm
def Perceptron(X, y):
    vectors=[]
    #Initialise weight vector to zero
    theta = np.zeros(2)
    misclassified = True
    while misclassified:
        misclassified = False
        for i in range(len(X)):
            if y[i] * np.dot(theta, X[i]) < 0:
                theta += y[i] * X[i]
                vectors.append(theta.tolist())
                misclassified = True
    return vectors
```

A.2 Implementation of the Greedy Perceptron

```
import numpy as np

# Find the loss given a weight vector
def findLoss(X,y,theta):
    loss = 0
    for i in range(len(X)):
```

```
            if y[i] * np.dot(theta, X[i]) <= 0:
                loss += 1
        return loss

# Calculate the loss after an update using different learning rates
def etasToLoss(X, y, theta, etas, i):
    losses = [findLoss(X,y, theta + y[i] * eta * X[i]) \
                        for eta in etas]
    min_index = np.argmin(losses)
    optimal_eta = etas[min_index]
    return optimal_eta

# Greedy Perceptron Algorithm
def GreedyPerceptron(X, y, etas):
    eta_list = []
    vectors = []
    theta = np.zeros(2)
    misclassified = True
    while misclassified:
        misclassified = False
        for i in range(len(X)):
            if y[i] * np.dot(theta, X[i]) < 0:
                eta = etasToLoss(X, y, theta, etas, i)
                eta_list.append(eta)
                theta += y[i] * eta * X[i]
                vectors.append(theta.tolist())
                misclassified = True
    return vectors
```

A.3 Code for Figure 3.4

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
import random
# Set the seed
np.random.seed(9758)
# Generate the dataset
X, y = datasets.make_blobs(n_samples=80, n_features=2, \
```

```
centers=2, cluster_std=1.5)
# Map 0 to 1 in the classification array
y = np.where(y==0,-1,1)

# Perceptron with fixed learning rate
def Perceptron(X, y, eta):
    vectors=[]
    #Initialise weight vector to zero
    theta = np.zeros(2)
    misclassified = True
    while misclassified:
        misclassified = False
        for i in range(len(X)):
            if y[i] * np.dot(theta, X[i]) < 0:
                theta += eta * y[i] * X[i]
                vectors.append(theta.tolist())
                misclassified = True
    return vectors

def plotData(X, y):
    fig, ax = plt.subplots()
    plt.plot(X[:,0][y==-1], X[:,1][y==-1], 'r^',markersize=2)
    plt.plot(X[:,0][y==1], X[:,1][y==1],'bs', markersize=2)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Mini Batch Classification Data with 2 classes')
    return

# Lists of vectors for Perceptron with different learning rates
vectors1 = Perceptron(X, y, 0.5)
vectors2 = Perceptron(X, y, 1)
vectors3 = Perceptron(X, y, 2)
vectors4 = Perceptron(X, y, 4)
# Vectors for Greedy Perceptron
greedy_vectors = GreedyPerceptron(X, y, [0.5, 1, 2, 4])

# Calculate the loss for each list of vectors
loss1 = [findLoss(X, y, vec) for vec in vectors1
loss2 = [findLoss(X, y, vec) for vec in vectors2]
```

```
loss3 = [findLoss(X, y, vec) for vec in vectors3]
loss4 = [findLoss(X, y, vec) for vec in vectors4]
greedy_loss = [findLoss(X, y, vec) for vec in greedy_vectors]

# Calculate the length of the longest list of vectors
lengths = [len(loss1), len(loss2), len(loss3), len(loss4), \
           len(greedy_loss)]
max_length = max(lengths)

# Create a list from 0 to max_length
iters = [i for i in range(1, max_length+1)]

# Extend the lists so that they are all of the same length
loss1x = loss1 + [0 for j in range(max_length - len(loss1))]
loss2x = loss2 + [0 for j in range(max_length - len(loss2))]
loss3x = loss3 + [0 for j in range(max_length - len(loss3))]
loss4x = loss4 + [0 for j in range(max_length - len(loss4))]
greedy_lossx = greedy_loss + [0 for j in range(max_length \
                                - len(greedy_loss))]
loss_lists = [loss1x, loss2x, loss3x, loss4x, greedy_lossx]

def plotLosses(iters, losses):
    for i in range(len(losses)):
        plt.plot(iters, l[i])
    # Add title and axes labels
    plt.title('Loss of Different Learning Rates \
              (including Greedy)')
    plt.xlabel('Iterations')
    plt.ylabel('Total Loss')

    rates = [1, 2, 3, 4, 'Greedy']
    plt.legend(rates, title = 'Learning Rates')

    plt.plot(len(loss1), 0, marker='o', markersize=6, \
    markerfacecolor = 'blue', markeredgecolor = 'blue')
    plt.plot(len(loss2), 0, marker='o', markersize=6, \
    markerfacecolor  = 'orange', markeredgecolor = 'orange')
    plt.plot(len(loss3), 0, marker='o', markersize=6, \
    markerfacecolor = 'green', markeredgecolor = 'green')
```

```
    plt.plot(len(loss4), 0, marker='o', markersize=6, \
    markerfacecolor = 'red', markeredgecolor = 'red')
    plt.plot(len(greedy_loss), 0, marker='o', markersize=6, \
    markerfacecolor = 'purple', markeredgecolor = 'purple')
    plt.show()

plotLosses(iters, loss_lists)
```