    * $\mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow$ predict $y = 1$
    * $\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow$ predict $y = -1$

- What can they express? What can they not express?
  - Expressive hypothesis class
    * Many functions are linear
    * Often a good guess for a hypothesis space
    * Some functions are not linear (i.e. `XOR`, non-trivial boolean functions)
    * However there are ways tof making them linear in a higher dimensional feature space
- Geometry
  - We can view the linear classifier as defining a *hyperplane* seperating instances in the answer space.
  - *Bias term* is needed because if b is zero, then restricting the learner only to hyperplanes that go through the origin which may not be expressive enough
- Feature expansion to predict a broader set of functions
  - *Forced Linearity* allows us to linearly classify non-linear data. For example we can take data to a higher dimension and perform linear classification in the raised dimension e.g. our instance space x can be brought paired with a polynomial making our instance space (x, x^2)
- Gradient Descent
  - Goal is to predict a real valued output using a feature representation of the input. We assume the output is a linear function of the inputs.
  - Learning is done by minimizing the total cost or loss function. Many algorithms in machine learning (perceptron ect...) follow this paradigm with different loss functions and different hypothesis space.
  - Gradient decent uses the bellow loss function
    * $J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x_i})^2$

Mistake Bound Learning

- One way of asking how good is your classifier
  - words
- The general structure of an online learning algorithm
  - words
- Goal: Counting Mistakes. What is a mistake bound algorithm
  - a mistake bound, or error driven, algorithm only makes updates when a prediction is incorrect. The weight vector is only altered in the case a mistake is made.
  - mistake bound algorithm is a algorithm that will acheive a desired result after a reasonable amount of corrections due to mistakes.
  - The Perceptron Convergence Theorem states that, If there exists a set of weights that are amenable to treatment with Perceptron (i.e., the data is linearly separable), then the Perceptron learning algorithm will converge
- Halving algorithm
  - words
- Perceptron algorithm, geometry of the update, margin, Novikoff's theorem, variants
  - The number of mistakes made by the perceptron algorithm is bound by the dimensionality R and the margin γ. γ defines the seperability of the data and is defined as the distance to the two nearest points in the positive and negative groupings of the instance space. The total number of mistakes is defined by $(\frac{R}{n})^2$. For booleans $R^2 = n$ since the L^2 norm of an n dimenstional vector is $\sqrt{n}$.

    * $\mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow$ predict $y = 1$
    * $\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow$ predict $y = -1$

- What can they express? What can they not express?
  - Expressive hypothesis class
    * Many functions are linear
    * Often a good guess for a hypothesis space
    * Some functions are not lineare (i.e. `XOR`, non-trivial boolean functions)
    * However there are ways tof making them linear in a higher dimensional feature space
- Geometry
  - *Bias term* is needed because if b is zero, then restricting the learner only to hyperplanes that go through the origin which may not be expressive enough
- Feature expansion to predict a broader set of functions
  - words

Mistake Bound Learning

- One way of asking how good is your classifier
  - **Protocol I**: The learner proposes instances as queries to the teacher
  - **Protocol II**: The teacher (who knows f) provides training examples
  - **Protocol III**: Some random source (e.g. Nature) provides training examples; the Teacher (Nature) provides the labels $(f(x))$
  - There are 100 boolean variables, but it is not known that only five are relevant: $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$. Want to know how many examples are needed to learn it.
    * Since we know that it's a monotone conjunction, it would take $n = 100$ queries for the straightforward approach to produce the hidden conjunction exactly.
    * Teacher gives the answer $\langle(0, 1, 1, 1, 1, 0, \cdots, 0, 1), 1\rangle$
    * Then the teacher proves that these variables are required by turning each one off and showing that the function produces 0
    * This straightforward algorithm has the function learned in $k = 6$ examples to produce the hidden conjunction exactly
  - If given a list of examples that are right and wrong from nature, you can just do a bitwise and operation on them to get the variables that are required. If this returns more functions than are needed, it doesn't matter since those variables are not deterministic for the function.
- The general structure of an online learning algorithm
  - Mistake Bound Alogrithms:
    * Setting:
      - Instance space: $\mathbf{X}$ (dimensionality n)
      - Target f: $\mathbf{X} \to \{0, 1\}$, $f \in C$, the concept class (parameterized by n)
    * Learning Protocol:
      - Learner is given $\mathbf{x} \in \mathbf{X}$, randomly chosen
      - Learner predicts $h(\mathbf{x})$, and is then given $f(\mathbf{x})$ (feedback)
    * Performance: Learner makes a mistake when $h(\mathbf{x}) \neq f(\mathbf{x})$
      - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f
      - $M_A(C) = \max_{f \in C, S} M_A(f, S)$: The maximum possible number of mistakes made by A for any target function C and any sequence S of examples
    * Algorithm A is a *mistake bound algorithm* for the concept class C if $M_A(C)$ is a polynomial in n (ie $\mathcal{O}(n)$)
  - No assumptions are made about the distribution of examples
  - Examples are presented teo the learning algorithm in a sequence. *Could be adversarial!*

  For each example:

- **Geometry** For a perceptron, the decision bound- ary is precisely where the sign of the activation, a, changes from 1 to +1. In other words, it is the set of points x that achieve zero ac- tivation. The points that are not clearly positive nor negative. For simplicity, well first consider the case where there is no bias term (or, equivalently, the bias is zero). Formally, the decision boundary B is:

  $x : \sum_d w_d x_d = 0$

  The sum is the dot product between the weight vector and x. This dot product is zero if the two vectors are perpendicular. The boundry is the perpendicular plane to w.
- **margin** Formally, given a data set D, a weight vector w and bias b, the margin of w, b on D is defined as:

  $margin(D, w, b) = min_{(x,y)\in D} y(w \cdot x + b)$ if w seperates D, $-\infty$ oherwise

  The margin on a data set is the largest obtainable margin, i.e. the supremum of the above.

- Winnow algorithm, mistake bound, balanced winnow
  - Mistake bound of the winnow algorithm for k-disjunctions is $O(klogn)$
  - to describe OR of r variables where r ¡¡ n takes O(rlog n) bits.
  - Winnow **mistake bound** is O(r log n)
  - Winnow learns the class of disjunctions in at most $2 + 3r(1 + logn)$ mistakes
  - the margin is $\gamma = \alpha / L_1(w^*) L_\infty(X))$ with bound $O((1/(\gamma^2 * logn)))$ **??**
- Perceptron vs. Winnow
  - The perceptron algorithm does addative updates. The Winnow algorithm does multiplicative. The perceptron mistake bound for k-disjunction is $O(n)$. The winnow for k-disjunctions is $O(klogn)$. Proof?
  - Use Winnow for multiplicative algorithms: If you believe that the hidden target function is sparce. Use Perceptron for addative functions: if the hidden target function is dense.
  - **Voted Perceptron** One way of using the perceptron is to award classifiers if they are succesful for a prolonged period of time before an update. To do this we must add a weight to succesful weight vectors. we therfore add a count to each success of a given classifier. If one classifier has 100 succesful classifications we add a weight of 100, incrementing this weight during each training example.

    $y = sign(\sum_{i=1}^{m} c^{(i)} sign(w^i \cdot x + b^i))$

    Although successful this method is insuficient in that it requires you store a mass of weighted weight vectors.
  - More practicle is the **average perceptron** which rather than voting on each training example we maintane a running sum of the averaged weight vectors and average bias

    $y = sign(\sum_{i=1}^{m} c^{(i)} w^i \cdot x + \sum_{i=1}^{m} c^{(i)} c^i b^i)$

  - **varient bounds?**

Batch Learning

- Assumption that train and test examples are drawn from the same distribution
  - Goal of batch learning: To devise good learning algorithms that avoid overfitting, namely, find a hypothesis that has a low chance of making a mistake on a new example.
  - Examples are drawn from fixed and maybe unknown probability distribution D.
  - Learning uses a training set S subset of D.
- How it is different from mistake bound learning
  - Online learning has no assumption about the distribution of the examples. Batch assumes there exists some probability distribution.
  - Online learning is done over a sequence of trials: learner sees an example, makes a prediction, and updates hypothesis based on true label. Batch learning is done over subset of the probability distribution.
  - Goal of online learning is to bound the number of mistakes whereas batch hopes to lower the probability of making a mistake.

  1. Learner gets an unlabeled example
  2. Learner makes a prediction
  3. Then, the true label is revealed
  - Count the number of mistakes
  - A concept class is learnable in the *mistake bound model* if there exists an algorithm that makes a polynomial number of mistakes for any sequence of examples
    - Polynomial in the size of examples
  - Important in the case of very large data sets, when the data cannot fit in memory
- Goal: Counting Mistakes. What is a mistake bound algorithm
  - Under the mistake bound model, we are not concerned about the number of examples needed to learn a function, only about not making mistakes.
  - Can be given the *same example* over and over, which under the mistake bound model is ok, since won't be able to learn the function **but** won't be making mistakes either.
  - General Mistake Bound Algorithm:
    - Let C be a finite concept class
    - Goal: Learn $f \in C$
    - Algorithm `CON`:
      - In the i^{th} stage of the algorithm:
      - C_i all concepts in C consistent with all $i-1$ previously seen examples
      - Choose randomly $f \in C_i$ and use to predict the next example
    - Clearly, $C_{i+1} \subseteq C_i$
    - If a mistake is made n the i^{th} example, then $|C_{i+1}| < |C_i|$ so progress is made
    - The `CON` algorithm mades at most $|C| - 1$ mistakes
- Halving algorithm
  - Let C be a finite concept class
  - Goal: To learn a diffen function $f \in C$
    - Initialize $C_0 = C$, the set of all possible functions
    - When an example $\mathbf{x}$ arives:
      - Predict the label for $\mathbf{x}$ as 1 if a *majority* of the functions in C_i predict 1. Otherwise 0. That is, output =1 if
        $$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$
    - If prediction $\neq f(\mathbf{x})$:
      - Update C_{i+1} = all elements of C_i that agree with $f(\mathbf{x})$
    - Learning ends when there is only one element in C_i
  - Proof:

    Suppose it makes n mistakes. Finally, we will have the final set of concepts C_n with one element

    C_n was created when a majority of the functions in C_{n-1} were incorrect, therefore

    $$\begin{aligned} 1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\ &< \frac{1}{2} \cdot \frac{1}{2}|C_{n-2}| \\ &< \vdots \\ &< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C| \\ &\Rightarrow \mathcal{O}(\log_2 |C|) \end{aligned}$$

- Perceptron algorithm, geometry of the update, margin, Novikoff's theorem, variants