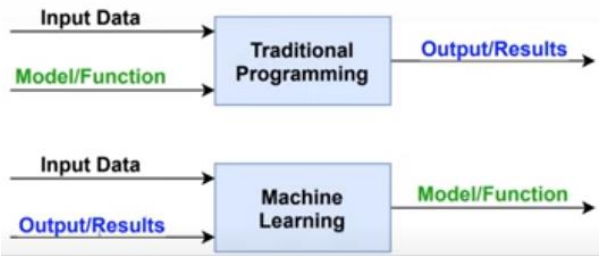
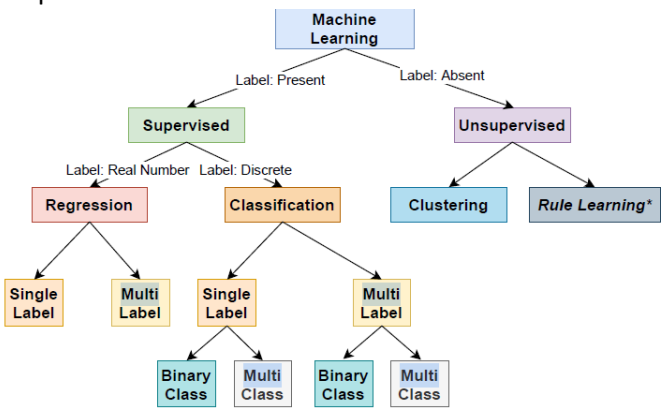
Week1

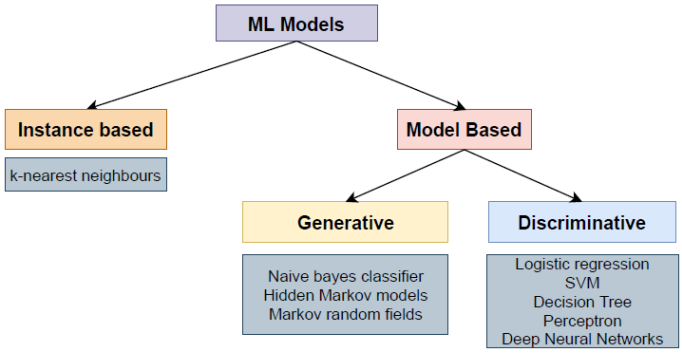
- Machine learning is subset of AI.
- Supervised and Unsupervised learning are two broad classes of machine learning algorithms.
- Linear Regression, Logistic Regression, Decision Trees, Support Vector Machines, Neural Networks are examples of supervised learning, while K-means clustering is unsupervised.
- Training Data, Model, Loss function, Optimization, Evaluation are the 5 steps following by all machine learning algorithms.
- Traditional programming works on inputs and pre-defined model/function, and generates outputs. Machine learning works on inputs and outputs, and generates models/function that fits. Once the model/function has been learnt, traditional programming can be employed to make predictions (generate outputs)



- Broad categorization of machine learning algorithms based on data. Typically, multi-class/single label classification problems are simply called *multi-class*, where are multi-class/multi-label classification problems are simply called *multi-label*. In the case of regression, multi-label problems are called multi-output.



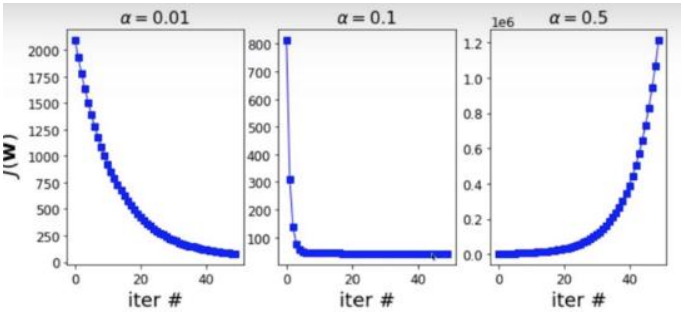
- Broad categorization of machine learning algorithms based on model.



- Machine learning algorithms are categorized into Batch learning and Online learning, based on learning style.
- **D** typically represents the set of training examples, a set of ordered pairs of (features, labels). $\boldsymbol{D} = \{(x^{(i)}, y^{(i)})\}$
- In a vectorized representation, we use matrix **X** to represent the input. $\mathbf{X} = [feature^{(1)\,T}, feature^{(2)\,T}, \ldots, feature^{(n)\,T}]$ and use vector **y** or matrix **Y** to represent the labels.
- *Training data* is used for training the model, *validation data* is used for tuning hyper-parameters (like regularization rate), and *test data* is used to evaluate the model performance.
- In cross-validation, we have several (say, *k*) sets of training-test data. Use (k - 1) partitions for training and 1 for validation. In an extreme case, k = n, each example in its own partition. Finally, compute average of evaluation metrics across *k* training/validation sets.
- Other techniques employed during preparing the data pre-processing.
  - Features are normalized to ensure that the convergence occurs faster. Discrete attributes are converted to numbers using one-hot encoding, hashing or embeddings.
  - Relationships between features and labels are examined using appropriate visualization techniques, or statistical methods like chi-square tests.
  - Data is subjected to cleansing (missing values) before using.
  - Class-imbalance is removed, by performing oversampling of minority classes.
- Example of a linear model, where there are m features is
  Output = $weight_0 + weight_1 * feature_1 + weight_2 * feature_2 + \ldots + weight_m * feature_m$
- More generically, a model is mathematically represented as h_w: X -> Y.
- Weight parameters are estimated from the training data. The set of weight parameters is called a weight vector.
- Weight parameters learnt from the training data works well with real-world data, since both sets of data are assumed to originate from the same distribution.
- When the output is a real number, we choose regression models. When the output is a discrete quantity, we choose classification models.
- Loss function is used to measure the difference between the predicted label and the actual label $J(W) = \sum_{i=1}^{n} [predicted^{(i)} - actual^{(i)}]^2$
- Loss is a function of the weight vector. This is mathematically represented as *J: W -> R*
- Loss function is *convex* in the case of linear/non-linear regression, support vector machines, and *non-convex* in the case of neural networks.
- Optimizing the loss function yields the optimal set of weights for generating the output closest to the actual, with minimal loss. This is mathematically represented as $W = argmin_W J(W)$
- Best way to reach the minimal value of loss is by equating derivative of loss to 0, mathematically represented as $\frac{d}{dW} J(W) = 0$
- Hence, one of the techniques using for optimization is (batch) gradient descent, where weight vector is updated in multiple short iterations. This can be mathematically represented as $w_i[t+1] \longleftarrow w_i[t] - \alpha \cdot \frac{\partial J(w)}{\partial w_i[t]}$

In the above equation, [t+1] is the iteration that follow [t], α is the learning rate, and $\frac{\partial J(w)}{\partial w_i[t]}$ is the partial derivative of the loss with respect to the weight vector.

- To speed up the (batch) gradient descent process, we use mini-batch gradient descent, where we work on a small set of examples, instead of the entire dataset. In the case of stochastic gradient descent (SGD), we use a batch-size of 1. Batch size is chosen to be a power of 2, to optimize the disk read/write during computations.
- Learning curves are used to measure the efficacy of the learning process. It plots iterations on X-axis and loss on the Y-axis, and expects loss to steeply climb down. If the loss *isn't reducing*, reduce the learning rate α and retry. If the loss *isn't reducing enough* in each iteration, increase the learning rate α and retry. Look at learning curves plotted for different values of α, in a linear regression problem.

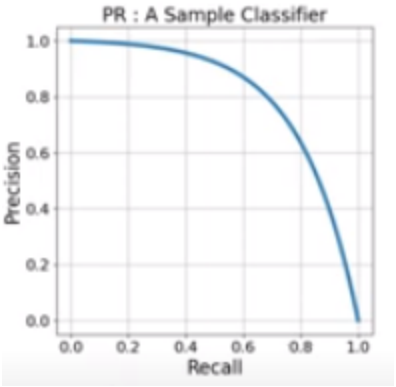


Note that in the first case (α=0.01), the loss reduces but not as quickly as in the second case (α=0.01). When α is higher at 0.5, the loss increases.

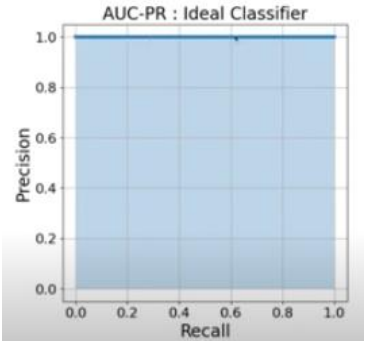
- When training and validation loss are low, it's the right fit. When training loss is low, and validation loss is high, it's an overfit. When the training and validation loss are high, it's an underfit.
- Overfitting can be avoided by learning from more data, or by using regularization (penalty)
- Underfitting can be avoided by increasing the model capacity by including the polynomial features, or by reducing regularization rate.
- Efficacy of the model can be measured by precision, recall, F1 score, AUC-ROC, AUC-PR. Accuracy isn't the best metric for measuring efficacy because it won't work well when there's class imbalance.
- *Confusion matrix* is used to calculate the above-mentioned metrics

Actual \ Predicted	False(0)	True(1)
False(0)	True Negative (TN)	False Positive (FP)
True(1)	False Negative (FN)	True Positive (TP)

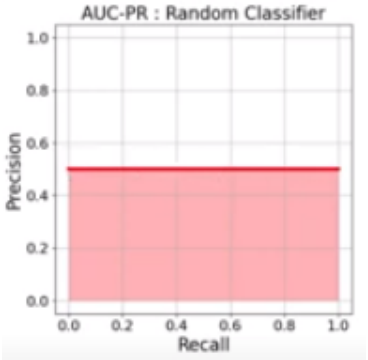
- Precision = TP / (TP + FP); Recall = TP / (TP + FN); F1-Score = 2 * Precision * Recall/(Precision + Recall);
- PR curve can be produced by plotting the precision-recall values at different thresholds of probability.
- A typical PR curve looks like this



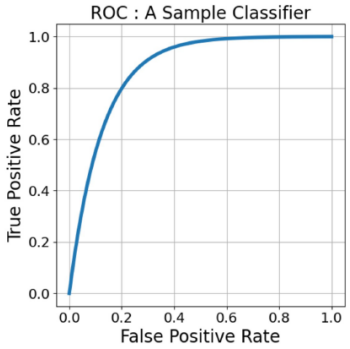
- PR curve for an ideal classifier looks like this.



- PR curve for a no-skills/random classifier with 0 class-imbalance looks like this. PR curve for a classifier with 70-30 class imbalance, has the line at 0.7 instead.



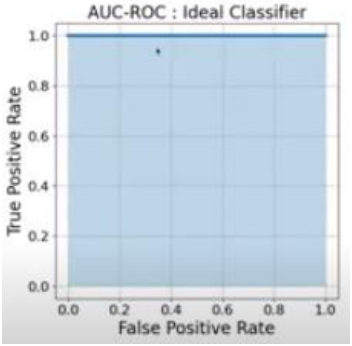
- AUC-PR is calculated by computing the total area under the PR curve. This is the preferred metric, when there's moderate to large class imbalance.
- ROC is plotted with False Positive Rate (FPR) on X-axis and True Positive Rate (TPR) on Y-axis, at different thresholds. Here's how it looks.



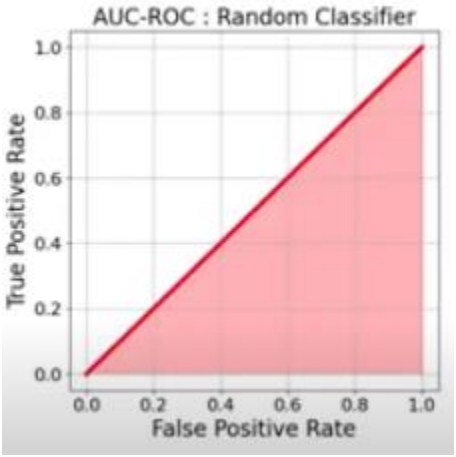
- Here's how FPR and TPR are calculated.

$$FPR = 1 - Specificity = \frac{FP}{FP + TN} \; ; \; TPR = Sensitivity = \frac{TP}{FN + TP}$$

- ROC curve for an ideal classifier looks like this



- ROC curve for a no-skills/random classifier looks like this, and has an area of 0.5. Anything below this area is worse than the random classifier. AUC-ROC is preferred as a metric when there's no class imbalance.



Week2

- In the case of linear regression, input is (n x m) feature matrix comprising of n examples each with m features. Label matrix is n x 1, where each label is a scalar.

- Linear regression model is represented mathematically as $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$, where the w_0, $w_1 \ldots w_m$ are the weights, and x_1, $x_2 \ldots x_m$ are the features. Since no feature is associated w_0, we add a dummy feature x_0 with value 1. This is compactly represented as $y = \sum_{i=0}^{m} w_i x_i$. In a vectorized form, this can be rewritten as $\mathbf{y} = w^T \mathbf{x}$. In the case of multiple examples, this can be rewritten as $\mathbf{y} = \mathbf{X}w$, where $\mathbf{X}$ is a matrix with shape *n x (m + 1)*, w has a shape *(m + 1) x 1* and **y** has a shape *n x 1*
- In the case of a single feature, the geometry of the model is that of a line; with 2 features, it's a plane; with >3 features, it's a hyper-plane with (m + 1) dimensions.
- Loss function (SSE) is represented mathematically as $J(W) = \sum_{i=1}^{n} [predicted^{(i)} - actual^{(i)}]^2$
- We usually divide the above value by a factor of 0.5, for mathematical convenience. This can be rewritten as $\frac{1}{2} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)} \right)^2$ or as (in vectorized notation) $\frac{1}{2} \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right)$
- Visualization of loss surface. As the yellow ball is dragged toward the bottom of the bowl-shaped loss surface, the predicted linear model matches the actual.
- Partial derivative of the loss is $\mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y})$. In the case of **normal equation** (fit), we'll equate this to zero, to give $w = X^{-1}y$.
- In the case of gradient descent, we'll start with an arbitrary weight vector, and keep updating the weights over multiple iterations (epochs). Weight update rule is given as $\mathbf{w}_{k+1} := \mathbf{w}_k - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ $:= \mathbf{w}_k - \alpha \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{Y})$, where α represents the learning rate. Note that the weight is updated with the product of (α, error, feature)
- Learning rate and number of iterations (epochs) are the two hyper-parameters.
- Lower learning rates takes much longer to reach the optimum loss, but very high rates could cause loss to increase. Use learning curves to find the optimum learning rate.
- Number of iterations must be high enough so that lowest loss could be reached, and must be low enough so that computational cycles aren't wasted after getting to the lowest loss.
- In Gradient descent algorithm, weights are updated simultaneously for all examples in the data. But, in mini-batch gradient descent, weights are updated simultaneously for a small subset of examples (in a mini-batch) from the data. In stochastic descent algorithm, weights are updated simultaneously only for single example at a time from the data. Thus, GD takes 1 iteration to update the weights for all examples, MBGD takes n/k iterations (where n is the total number of examples and k is the mini-batch size), and SGD takes n iterations.
- Evaluation is done using the root-mean square error (RMSE) measure. RMSE is calculated as $\sqrt{\frac{2}{n} \cdot J(w)}$

Week3

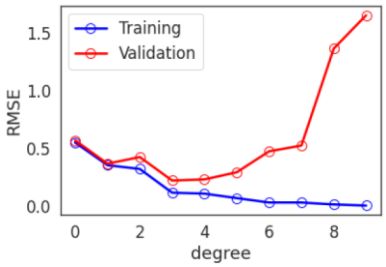
- When you fit polynomial data, say, generated by $\sin(2\pi x)$ using linear model, it underfits. In this case, we'll need to make a *polynomial transformation* of the single input feature x to the k^{th} degree, and

generate new features x^2, x^3, …x^k. Note that k is an arbitrary number. Once we've *k* features, we'll use these features to fit into a linear model. This will *closely approximate* the polynomial fit of the single original feature.

- Denoting these transformations to the input feature using the label ϕ, we can mathematically represent the fitment as $y = \mathbf{w}^T \phi(\mathbf{x})$
- Lower degree polynomial transformations tend to underfit, while higher degree tend to overfit.
- Here's a pictorial representation of the process of *generating* the polynomial features.

Input features	[[1,2], [3,4]]			Input features	[[2, 3], [4, 5]]	
Column-wise	[[1,3], [2,4]]			Column-wise	[[2,4], [3,5]]	
Combinations				Combinations		
Degree	Combinations	Dot Product		Degree	Combinations	Dot Product
0	([1,1])	[1,1]		0	([1,1])	[1,1]
1	([1,3])	[1,3]		1	([2,4])	[2,4]
1	([2,4])	[2,4]		1	([3,5])	[3,5]
2	([1,3], [1,3])	[1,9]		2	([2,4],[2,4])	[4,16]
2	([2,4],[2,4])	[4,16]		2	([3,5],[3,5])	[9,25]
2	([1,3], [2.4])	2, 12		2	([2,4],[3,5])	[6,20]
				3	([2,4],[2,4],[2,4])	[8,64]
				3	([3,5],[3,5],[3,5])	[27,125]
				3	([2,4],[3,5],[3,5])	[18,100]
				3	([3,5],[2,4],[2,4])	[12,80]

- RMSE vs degree plots are typically used to identify overfit/underfit tendency. Thus, in following graph, the model tends to overfit more than degree 6.



- In the case of higher degree polynomials, the weights tend to be arbitrarily large for the features. This is a symptom of overfit. This is also called high variance problem.
- In order to avoid overfit, use larger training sets, or use regularization to control the model complexity.
- Regularization process adds a penalty to the loss, leading to a change in its derivative, and hence the weight update.
- Regularization is controlled by two terms – the penalty function (function of weight vector) and the regularization rate.
- 3 types of regularization are used in the industry – L1 (Lasso), L2 (Ridge), and a combination of L1 and L2.
- Ridge regularization uses the second norm of the weight vector, and takes the following vectorized form

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Derivative of the loss after applying the ridge regularization is $\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}$.

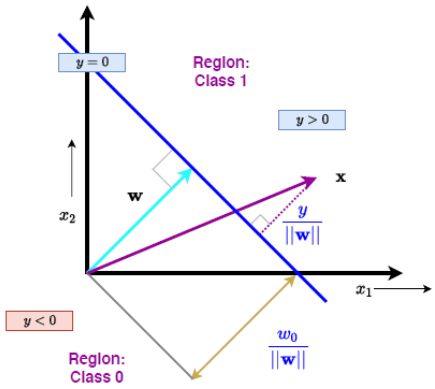
Equating this to 0, the normal equation takes the form $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

- In the case of gradient descent, we'll start with an arbitrary weight vector, and keep updating the weights over multiple iterations (epochs). Weight update rule is given as $\mathbf{w}_{k+1} := \mathbf{w}_k - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ $:= \mathbf{w}_k - \alpha \left\{ \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w} \right\}$, where α represents the learning rate.
- With increasing rates of regularization λ, the overfit reduces; but beyond a certain threshold of the λ, it causes the model to underfit.
- To find out the right value of λ, train the model and for each value of λ, calculate the cross-validation error (loss or rmse) on validation set. The regularization rate that gives the least cross-validation error is the right choice of λ.
- λ Vs cross-validation error typically takes a bowl shape when plotted. Most appropriate value of the λ is at the lowest point of the bowl.
- Lasso regularization uses the first norm of the weight vector, and takes the following form. $J(\mathbf{w}) = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2} \sum_{j=1}^{m} |w_j|$
- Since the above form is not differentiable, we use special methods (from sklearn library) to carry out this process.

Week4

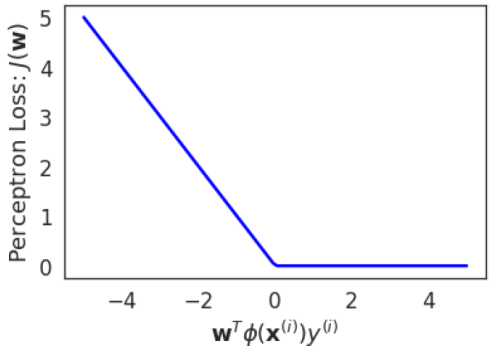
- Class label is a discrete quantity, unlike a real number in the regression setup.
- Under single-label classification (aka *binary/multi-class classification*), each example gets assigned with a single label, whereas under *multi-label classification*, each example gets assigned with multiple labels. Note that multi-class, multi-label problems are typically referred to simply as *multi-label* problems.
- Classification methods include
  - Discriminant functions that learn direct mapping between feature matrix x and labels.
  - Generative models that learn conditional probability distribution P(y|x) using Bayes theorem and prior probabilities of classes, and assign labels based on that.
  - Discriminative models that use parameters to build models based on conditional probability, and assign labels on that.
  - Instance-based models that compare training and test examples, and assign class labels based on certain measures of similarity.
- Multi-class and multi-label classifications use one-hot encoding to represent classes assigned to an example. While in the former case, exactly one entry in each row is 1, whereas in the latter, multiple entries on each row could be 1. In the case of binary classification, the classes are represented as -1 and +1 and doesn't use any encoding.
- In the case of multi-class classification, feature matrix X is represented by *(n x m)* matrix, labels by *(n x 1)* vector and weights by *(m x 1)* vector. In the case of multi-label classification (with k classes), feature matrix X is represented by *(n x m)* matrix, labels by *(n x k)* matrix and weights by *(m x k)* matrix.
- Discriminant function has the same mathematical representation as the linear regression $\mathbf{y = w_0 + w^T x}$, where $\mathbf{w_0}$ is the bias. It has the geometry of a hyper-plane with *(m-1)* dimensions, where *m* is the number of features.
- The decision boundary of the classes class-0 and class-1 is y = 0. When y > 0, it belongs to the class-1, else it belongs to the class-0.

- The weight vector is orthogonal to every vector lying within the decision surface; hence it determines the orientation of the decision surface. The location of the decision boundary is $w_0/||w||$ and y is the signed measure of the perpendicular distance of the point x from the decision surface. This is represented in the figure below.



- When there are multiple classes, we could use One-Vs-rest (where we build *k-1* discriminant functions) or one-Vs-one (where we build kC2 discriminant functions), both of which have their faults. Hence, we develop single k-class discriminant function that carries the form $y_k = w_{k0} + \mathbf{w_k}^T \mathbf{x}$. In the case of k-class discriminant function, the correct classification is done majority vote. Thus, label y_k is assigned to an example, if $y_k > y_j$ for all $j \neq k$
- To learn parameters of model, we may use *Least Squares Classification* or *Perceptron*.
- Calculations of loss, optimization and weight update rule in the case of *Least Squares Classification* remains the same as in the case of Linear Regression, which computes J(w) = $\frac{1}{2}(\mathbf{Xw} - \mathbf{y})^T(\mathbf{Xw} - \mathbf{y})$; w = $X^{-1}y$ is the normal equation; and $\mathbf{w}_{k+1} := \mathbf{w}_k - \alpha\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ $:= \mathbf{w}_k - \alpha\mathbf{X}^T(\mathbf{Xw} - \mathbf{Y})$ is the weight update rule. The equations used in the case of regularization also remain unchanged.
- From a code perspective, the only change required is to change the *predict* function from `return X @ self.w` to `return np.argmax(X @ self.w, axis=-1)`. However, note that the *loss* and *optimization* (*calculate_gradient* method) procedures use *predict_internal* method that continues to use `return X @ self.w`
- Perceptron, a basic classification algorithm was invented by Frank Rosenblatt in 1958.
- Perceptron can solve only binary classification problems, and hence label matrix has n x 2 shape. Labels can be {-1,+1}
- Perceptron model can be mathematically represented as $y = \text{sign}(\mathbf{w}^T\phi(\mathbf{x}))$, where w is the weight vector, and **X** is the transformed feature matrix**.** Hence, this is essentially a step function. More completely, $y = f(\mathbf{w}^T\phi(\mathbf{x}))$ is the equation used to represent the model, where *f* is a non-linear activation function. Note that $\mathbf{w}^T\phi(\mathbf{x})$ is sometimes written as $h_w(\mathbf{X})$

- Loss of each example in the perceptron model can be represented as $J(\mathbf{w}) = \sum_{i=1}^{n} \max(0, -y^{(i)} h_{\mathbf{w}}(x^{(i)}))$. Losses for all examples are summed up to get the cumulative loss; Loss function is not differentiable and can be geometrically represented as



- In the misclassified region, loss can be reduced by reducing the weight vector. In the correctly classified region, leave the weight vector unchanged.
- While linearly separable examples lead to zero loss ultimately and convergence of the algorithm, non-separable examples lead to oscillating loss and hence don't converge.
- Similar to the loss, the weight update rule for each example can be represented as $\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \left(y^{(i)} - \hat{y}^{(i)}\right) \phi(\mathbf{x}^{(i)})$, and must be summed up to get the cumulative weight update. This is similar to weight update rule in all models where weights are updated with the product of (α, error, feature).