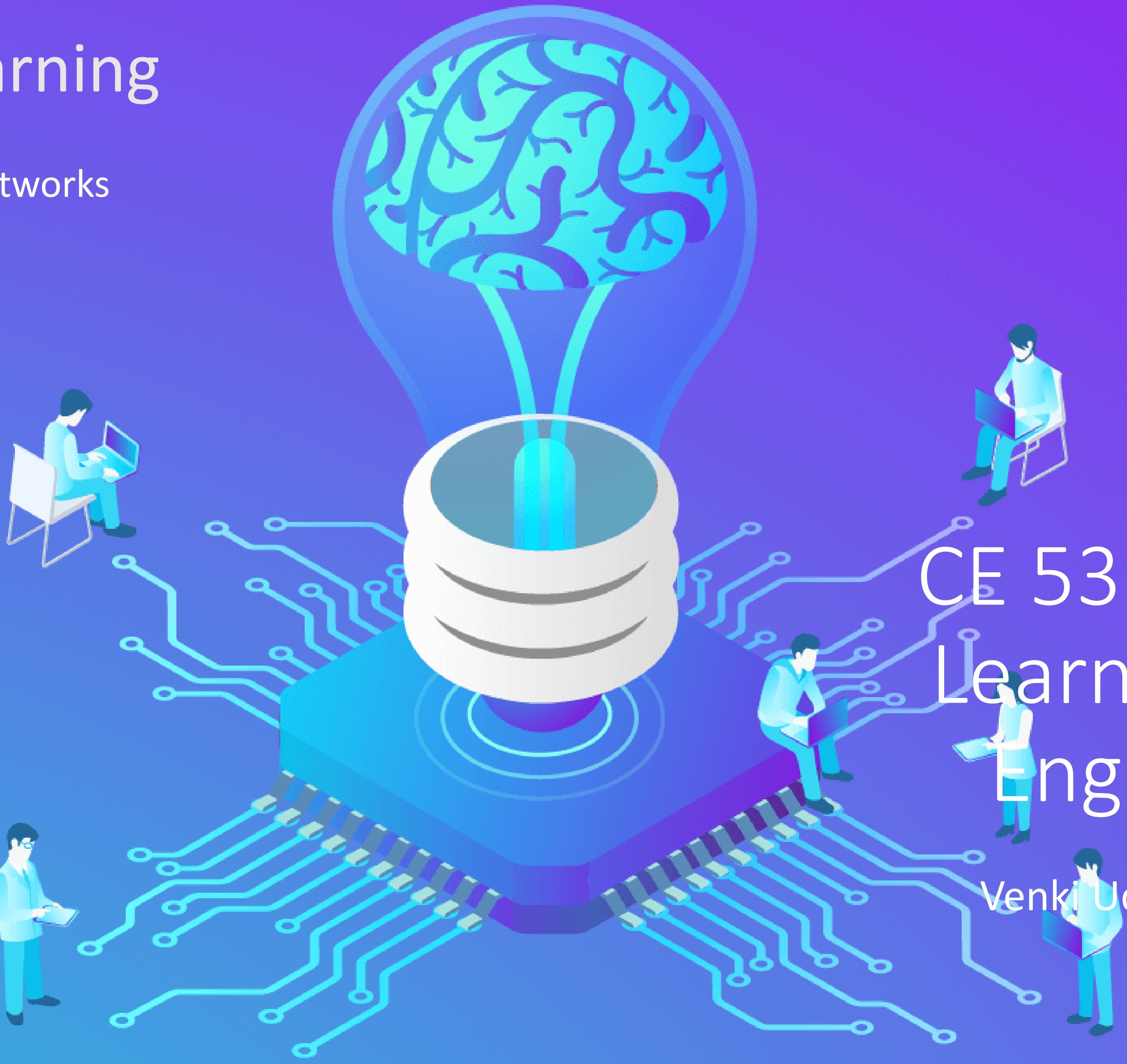
Machine Learning

Artificial Neural Networks
Perceptrons



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap

- What is Machine Learning
- How is it useful for Civil Engineers
- Overview of Machine Learning Methods
- Linear Regression
  - Bivariate
  - Regression interpretation
  - Multivariate
- Logistic Regression
  - Maximum likelihood estimation
  - Regularization (introduction)
- Naïve Bayesian Classifier
  - What is it
  - What makes it naïve
  - Bayes theorem
  - Prior, likelihood and posterior
- K-Nearest Neighbor
  - How does the algorithm work
  - Why is it a lazy learner
  - How to do regression and classification
- Introduction to Decision Trees
  - Fundamentals
  - Information Gain, Entropy and Gini Index
  - ID3 algorithm
  - Classification and Regression Trees (CART)
  - Multi-Adaptive Regression Splines (MARS)

- Ensemble learners
  - Introduction
- Their benefits and drawbacks
- Simple (voting) ensemble learners
- Bagging and Pasting
- Generic bagging classifiers
- Random Forest classifiers
- Boosting Classifier
  - Adaboost
- Unsupervised classification
  - K Means Learning

Python – Introduction
Python – Functions
Python - Pandas
Python – np, scipy, statsmodels
Python – Scikit learn – linear, metrics
Python – Matplotlib, seaborn
Python – Mixed_Naive_Bayes
Python – scikit learn neighbors module
Python – scikit learn ensemble voting
Python – scikit learn bagging classifier
Python – scikit learn RandomForestClassifer
Python – scikit learn AdaBoostClassifier
Python – scikit learn KMeans

R – Classification and Regression Trees using rpart
R – Drawing trees using rpart.plot
R - Multiadaptive Regression Splines (MARS) using Earth Algorithm

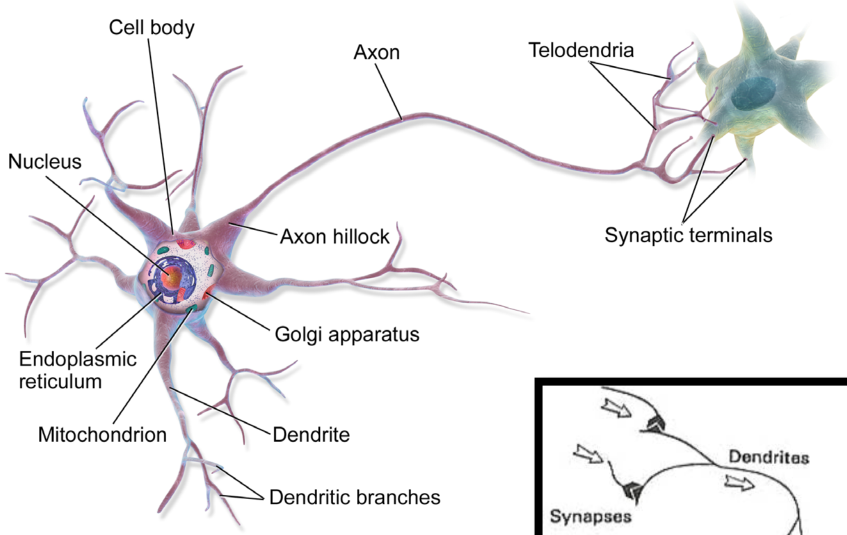
Artificial Neural Networks

Goals

- For the remainder of the course we shall be focused on Artificial Neural Networks (ANNs)
- Literature on ANNs is vast and there have been some exciting developments in the last few years
  - Deep Learners
  - Convolution Neural Networks
- ANNs and its variants have been used extensively in Civil Engineering
  - Applications of ANNs to civil engineering can be dated back to early 1990s
  - More so than most other Machine Learning Methods
  - Google Scholar search of Artificial Neural Networks + Civil Engineering results in over 133,000 publications
    - Likely even more undocumented studies
- We shall with the fundamentals and move to some more recent (interesting) applications
- We will make use of Keras and Tensor Flow (TF) packages for this part of the course

What are ANNs

- ANNs are algorithms that mimic the functioning of the brain
  - Mimic Human Neurons in general



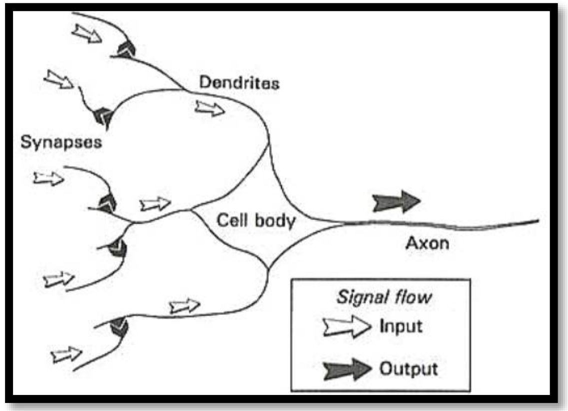
Synaptic terminals (synapses) of one neuron are connected to synaptic terminals of others

When there is a stimulus – Biological neurons produce an electrical impulses (action potential or signal)

Electrical signals travel through axons and make synapses release chemical signals (neurotransmitters)

When a neuron receives sufficient neurotransmitters it releases its own electrical impulse (it can also amplify or inhibit it in some cases)

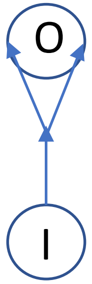
A human brain has 10^{11} neurons and each neuron is connected to 10^4 neurons



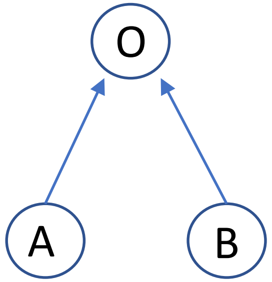
Signals are passed through layers of interconnected neurons
Output of a neuron is an electrical signal which is an input to another

Artificial Neurons

- McCulloh and Pitts (MP) proposed a simple model of the neuron in 1943
  - Receives one or more binary (on/off) inputs
  - Provides a binary (on/off) output
    - On output is provided when the majority of the inputs received are in the 'on' mode
- Even with this simplicity the artificial neuron can mimic several logical (Boolean) operations
  - AND, OR

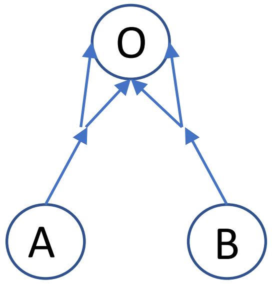


IF I is + then O is ++
(Result of O is a +)



IF A is + AND B is + Then O is ++ (Result of O is a +)

IF A is + AND B is - Then O is + (Result of O is a -; No Majority)



IF A is + OR B is + Then O is ++ (Result of O is a +)

Perceptron

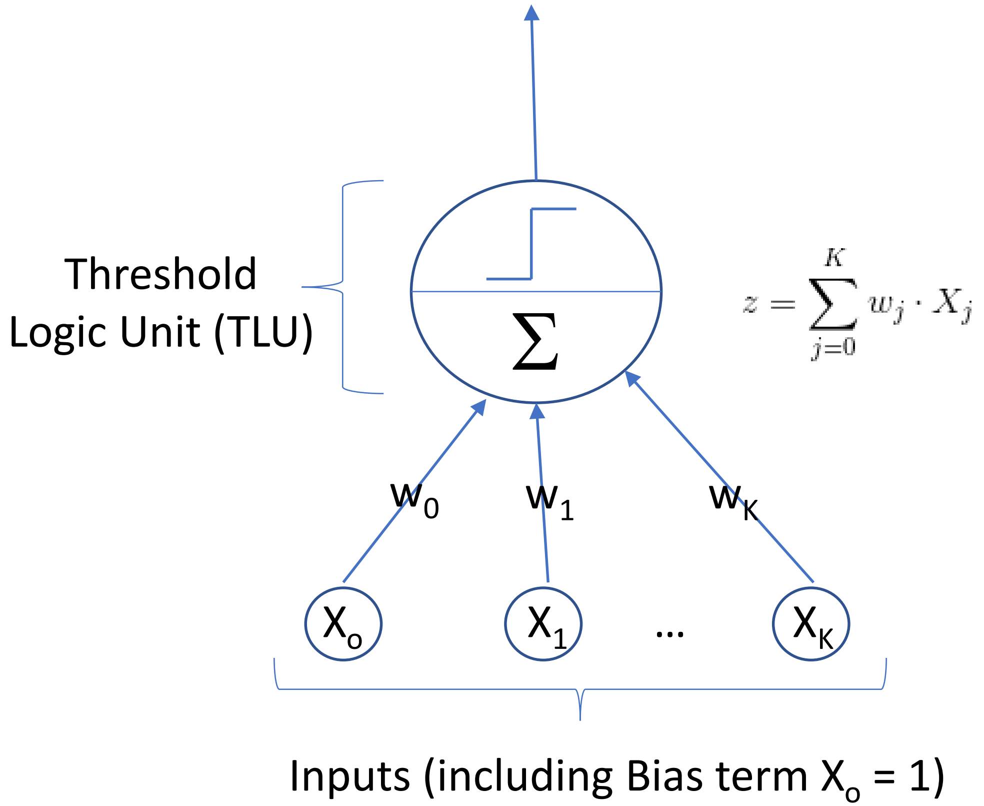
- An alternative model of Neuron proposed by Frank Rosenblatt in 1957
  - Inputs are numbers (instead of binary values)
  - Each input connection to the output has a connection weight
  - A weighted addition is performed first
  - The output of the weighted addition is used to determine the output state

Heaviside Step Function

$$Y = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Sign Function

$$Y = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z \geq 0 \end{cases}$$

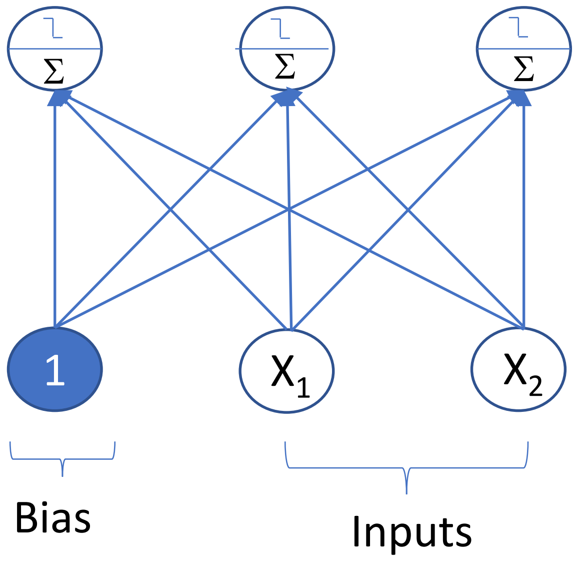


A single TLU can be used for binary classification

Unknown weights have to be obtained via optimization

Multi-output Perceptrons

- Inputs can be connected to multiple outputs
  - Multinomial classifiers



$$\overbrace{\begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} \\ z_{2,1} & z_{2,2} & z_{2,3} \\ \dots & \dots & \dots \\ z_{N,1} & z_{N,2} & z_{N,3} \end{bmatrix}}^{\text{Summation Matrix}} = \overbrace{\begin{bmatrix} 1 & X_{1,1} & X_{1,2} \\ 1 & X_{2,1} & X_{2,2} \\ \dots & \dots & \dots \\ 1 & X_{N,1} & X_{N,2} \end{bmatrix}}^{\text{Input Matrix}} \overbrace{\begin{bmatrix} w_{0,1} & w_{0,2} & w_{0,3} \\ w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{bmatrix}}^{\text{Weight Matrix}}$$

Instance TLU #

Bias Instance (data row) Attribute (data column)

Input (Data Column) TLU #

Once the summation matrix is obtained it can be sent through an appropriate threshold function (e.g., Heaviside Step Function)

Training Perceptrons

- Training Perceptrons implies identifying unknown weights
- Perceptrons are classically trained using Hebbian Learning
  - Neurons that fire together wire together
  - If a (biological) neuron triggers another neuron the connection between them grows stronger
- In Hebbian learning the connections that make correct predictions are reinforced
  - Weights are updated for those nodes that make incorrect predictions to improve their performance
  - Weights of nodes making correct predictions are left alone
    - Same weight carries for next step

Training Perceptrons

- The Hebbian Training procedure is as follows:
    1. Randomly initialize the connection weights
    2. Pass one instance through the perceptron
        A. Make a prediction
    3. Use the actual output and the predicted output to compute the error at each TLU
    4. Update the weights using the Hebbian Update formula
    5. Iterate step 2 – 4 by passing other instances
        1. The weight changes are small

Old Weight Estimate

New Weight Estimate

$$w_{i,j}^{t+1} = w_{i,j}^{t} + \eta \overbrace{(y_{o,j} - y_{p,j})}^{\text{Residual}}$$

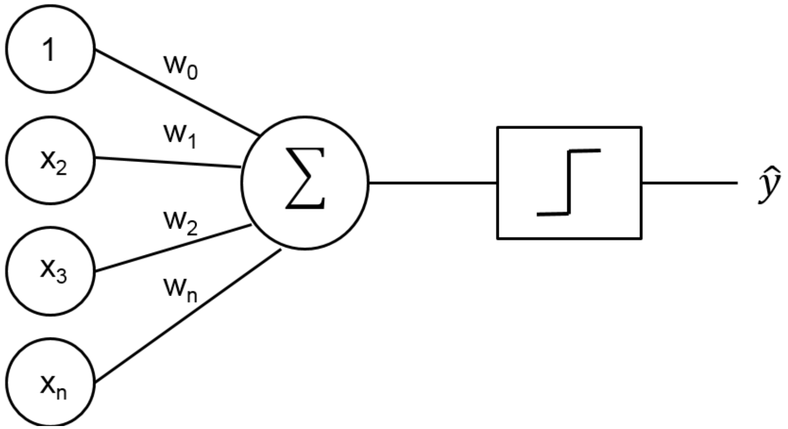
Learning Rate

Learning Rate is the step size for changing the weight

Hebbian training is very similar to the Gradient Descent algorithm

Perceptrons

- Perceptrons are linear classifiers
  - The decision boundary is linear
  - Similar to Logistic Regression
  - They cannot learn nonlinear relationships
- Perceptrons directly provide a binary output (Yes/No)
  - Unlike Logistic Regression which provided probability of a state



Illustrative Example

- Predicting damage to culverts in Texas
- Create Perceptron to classify satisfactory and unsatisfactory states

Satisfactory

Unsatisfactory

Code	Meaning	Description
9	Excellent	As new
8	Very Good	No problems noted.
7	Good	Some minor problems.
6	Satisfactory	Structural elements show some minor deterioration.
5	Fair	All primary structural elements are sound but may have minor section loss, cracking, spalling or scour.
4	Poor	Advanced section loss, deterioration, spalling or scour.
3	Serious	Loss of section, deterioration, spalling or scour has seriously affected primary structural components. Local failures are possible. Fatigue cracks in steel or shear cracks in concrete may be present.
2	Critical	Advanced deterioration of primary structural elements. Fatigue cracks in steel or shear cracks in concrete may be present or scour may have removed substructure support. Unless closely monitored it may be necessary to close the bridge until corrective action is taken.
1	Imminent Failure	Major deterioration or section loss present in critical structural components or obvious vertical or horizontal movement affecting structure stability. Bridge is closed to traffic but with corrective action may put back in light service.
0	Failed	Out of service, beyond corrective action.

Source: United States Department of Transportation. Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges. Washington, D.C., 1995, page 38.

Python Implementation

- Scikit learn package has a Perceptron function in its linear_model module
  - It actually uses stochastic gradient algorithm to find the unknown weights

Notes

`Perceptron` is a classification algorithm which shares the same underlying implementation with `SGDClassifier`. In fact, `Perceptron()` is equivalent to `SGDClassifier(loss="perceptron", eta0=1, learning_rate="constant", penalty=None)`.

`sklearn.linear_model`.Perceptron

class `sklearn.linear_model.Perceptron`(*penalty=None, alpha=0.0001, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False*) [source]

Read more in the User Guide.

Parameters:

penalty : *{'l2','l1','elasticnet'}, default=None*
The penalty (aka regularization term) to be used.

alpha : *float, default=0.0001*
Constant that multiplies the regularization term if regularization is used.

fit_intercept : *bool, default=True*
Whether the intercept should be estimated or not. If False, the data is assumed to be already centered.

max_iter : *int, default=1000*
The maximum number of passes over the training data (aka epochs). It only impacts the behavior in the `fit` method, and not the `partial_fit` method.

New in version 0.19.

tol : *float, default=1e-3*
The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol).

New in version 0.19.

shuffle : *bool, default=True*
Whether or not the training data should be shuffled after each epoch.

verbose : *int, default=0*
The verbosity level

eta0 : *double, default=1*
Constant by which the updates are multiplied.

n_jobs : *int, default=None*
The number of CPUs to use to do the OVA (One Versus All, for multi-class problems) computation. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

random_state : *int, RandomState instance, default=None*
The seed of the pseudo random number generator to use when shuffling the data. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`.

early_stopping : *bool, default=False*
Whether to use early stopping to terminate training when validation. score is not improving. If set to True, it will automatically set aside a stratified fraction of training data as validation and terminate training when validation score is not improving by at least tol for n_iter_no_change consecutive epochs.

New in version 0.20.

validation_fraction : *float, default=0.1*
The proportion of training data to set aside as validation set for early stopping. Must be between 0 and 1. Only used if early_stopping is True.

New in version 0.20.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

Python Code

```
# Step 1: Import Libraries
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn import metrics
import seaborn as sns
from matplotlib import pyplot as plt
```

```
# Step 2: Change working directory
dir = 'D:\\Dropbox\\000CE5333Machine Learning\\Week9ANNS\\Code'
os.chdir(dir)

# Step 3: Read the dataset
a = pd.read_csv('TXculvertdata.csv') # read our dataset
features = ['SVCYR','ADT','Reconst','PTRUCK'] # INPUT DATA FEATURES
X = a[features] # DATAFRAME OF INPUT FEATURES
SVCYR2 = a['SVCYR']**2  # Add SVCYR square to the dataset
X['SVCYR2'] = SVCYR2  # CALCULATE THE SQUARE OF AGE
Y = a['Culvert_Damage'] # ADD IT TO THE INPUT FEATURE DATAFRAME
```

```
# Step 4: Split into training and testing data
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,
                    random_state=10)
```

```
# Step 5: Fit the perceptron model
prcp = Perceptron()
prcp.fit(X_train,y_train)
```

```
# Step 6: Make Predictions and compute confusion matrix
y_pred=prcp.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix # y_test is going be rows (obs), y_pred (predicted) are cols
```

```
# Step 7: Evaluate usng accuracy, precision, recall
print("Accuracy:",metrics.accuracy_score(y_test, y_pred)) # overall accuracy
print("Precision:",metrics.precision_score(y_test, y_pred)) # predicting 0 (Sat)
print("Recall:",metrics.recall_score(y_test, y_pred)) # predicting 1 (unsat)
```

Results

- The Perceptron model does not do a good job compared to most other models

Model	Accuracy
Logistic	0.6657
Naïve Bayes	0.6557
KNN	0.6236
Ensemble	0.6676
Perceptron	0.5611

Perceptron

Precision	0.4711
Recall	0.4792

Perceptron Coefficients

Parameter	**Coefficient**
Intercept	11144.
SVCYR	-151405.
ADT	907.
Reconst	908.
PTRUCK	-104361.
SVCYR2	2165.

Perceptron

	Predicted	
Obs	0	1
0	2170	1335
1	1292	1189

LR Model

	Predicted	
Obs	0	1
0	2716	789
1	1230	1251

Unlike LR – Perceptron does not give probabilities so a ROC cannot be constructed

You Should Know

- What are biological neurons
  - How do they function?
- The McCulloh-Pitt (M-P) artificial neuron
  - How do they work
- Perceptron artificial neuron
  - Linear classifier
- Hebbian learning
- How are the weights updated
- Limitations of Perceptrons
  - Linear classifier