

Q:1 Write the differences between Certainty and Probability.

Ans. The exact point at which one ceases to be certain is the degree of certainty as opposed to the degree of belief measured as a probability function. Probability is quantified as a number between 0 and 1 (where 0 indicates impossibility and 1 indicates certainty). The higher the probability of an event, the more certain we are that the event will occur. The words “certainty” and “probability” do not apply to propositions that are either true or false. These propositions entertained by us with suspended judgment should never be qualified as either certain or probable. In American common law, there are degrees of certainty and doubt. Certainty attaches to judgments beyond the shadow of doubt; not certain are judgments made with a reasonable doubt, and less certain still are judgments made by a preponderance of the evidence. The last two are judgments to which some degree of probability must be attached, the former more probable, the latter less probable. The propositions in each of these two cases, when entertained with suspended judgment, are either true or false. Certainty and probability qualify our judgments about the matters under consideration on the propositions entertained with suspended judgment. This statement brings us to consider what happens by chance and what is causally determined. Here we must distinguish between the mathematical theory of probability and the philosophical theory of what happens by chance.

Q:2 Difference between complete and admissible algorithm.

Ans:

- Completeness: An algorithm is complete if it terminates with a solution when one exists.
- Admissibility: An algorithm is admissible if it is guaranteed to return an optimal solution whenever a solution exists.

Depth-First Search

Completeness: not guaranteed in general
Admissibility: not guaranteed.

Breadth-First Search

Completeness: guaranteed.
Admissibility: the algorithm will always find the shortest path

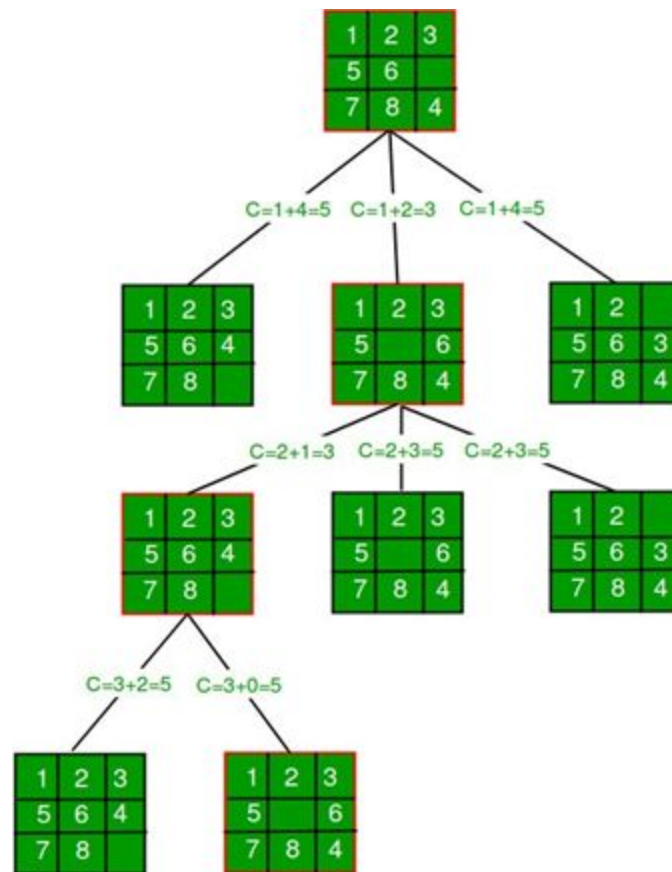
**Depth-First Search
Iterative-Deepening**

Completeness: guaranteed.
Admissibility: the algorithm will always find the shortest path

Q:3 Compute the complexity of the 8 puzzle problem.

Ans: Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

In this solution, successive moves can take us away from the goal rather than bringing closer. The search of state space tree follows leftmost path from the root regardless of initial state. An answer node may never be found in this approach.



Time Complexity= 9!

Q:4 Write the algorithm for IDFS.

Ans: function IDFS(root)

```
    for depth from 0 to  $\infty$ 
        found, remaining <- DLS(root,depth)
        if found  $\neq$  null
            return found
        else if not remaining
            return null
function DLS(node,depth)
    if depth= 0
        if node is goal
            return (node,true)
        else
            return(null,true)
    else if depth>0
        any_remaining <- false
        foreach child of node
            found, remaining <- DLS(child,depth-1)
            if found  $\neq$  null
                return(found,true)
            if remaining
                any_remaining <- true
        return (null,any_remaining)
```

Q:5 Write the algorithm for bidirectional search.

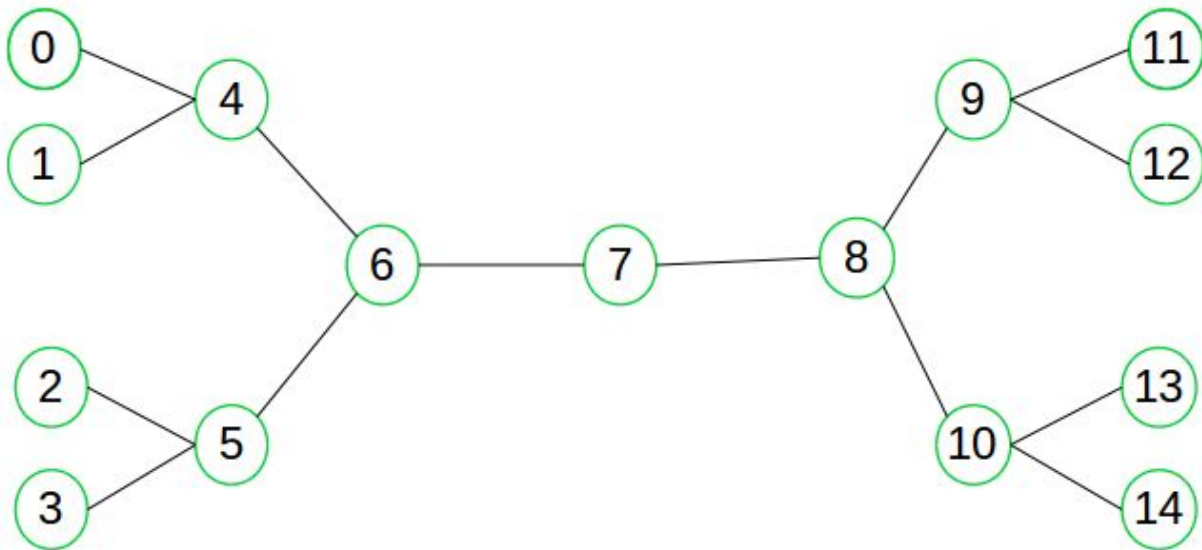
Ans: Bidirectional search is a graph search algorithm which find shortest path from source to goal vertex. It runs two simultaneous search –

1. Forward search from source/initial vertex toward goal vertex
2. Backward search from goal/target vertex toward source vertex

Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs – one starting from initial vertex and other starting from goal vertex.

The search terminates when two graphs intersect.

Just like [A* algorithm](#), bidirectional search can be guided by a [heuristic](#) estimate of remaining distance from source to goal and vice versa for finding shortest path possible. Consider following simple example-



Suppose we want to find if there exists a path from vertex 0 to vertex 14. Here we can execute two searches, one from vertex 0 and other from vertex 14. When both forward and backward search meet at vertex 7, we know that we have found a path from node 0 to 14 and search can be terminated now. We can clearly see that we have successfully avoided unnecessary exploration.

Why bidirectional approach?

Because in many cases it is faster, it dramatically reduce the amount of required exploration. Suppose if branching factor of tree is **b** and distance of goal vertex from source is **d**, then the normal BFS/DFS searching complexity would be $O(b^d)$. On the other hand, if we execute two search operation then the complexity would be $O(b^{d/2})$ for each search and total complexity would be $O(b^{d/2} + b^{d/2})$ which is far less than $O(b^d)$

When to use bidirectional approach?

We can consider bidirectional approach when-

1. Both initial and goal states are unique and completely defined.
2. The branching factor is exactly the same in both directions.

Performance measures

- Completeness : Bidirectional search is complete if BFS is used in both searches.
- Optimality : It is optimal if BFS is used for search and paths have uniform cost.

- Time and Space Complexity : Time and space complexity is $O(b^{d/2})$

Q:6 Write the algorithm for optimal A* search technique.

Ans:

1. Initialize the open list
2. Initialize the closed list
 - put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty
 - a) find the node with the least f on the open list, call it "q"
 - b) pop q off the open list
 - c) generate q's 8 successors and set their parents to q
 - d) for each successor
 - i) if successor is the goal, stop search
 - $$\text{successor.g} = \text{q.g} + \text{distance between successor and q}$$

$$\text{successor.h} = \text{distance from goal to successor}$$
 (This can be done using many ways, we will discuss three heuristics- Manhattan, Diagonal and Euclidean Heuristics)

$$\text{successor.f} = \text{successor.g} + \text{successor.h}$$
 - ii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor
 - iii) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor

otherwise, add the node to the open list
end (for loop)

e) push q on the closed list
end (while loop)

Q:7 Explain the algorithm for IDA*.

Ans:

root=initial node;

Goal=final node;

function IDA*()

{

threshold=heuristic(Start);

while(1)

{

integer temp=search(Start,0,threshold); //function search(node,g score,threshold)

if(temp==FOUND)

return FOUND;

if(temp== ∞)

return;

threshold=temp;

}

}

function Search(node, g, threshold)

{

f=g+heuristic(node);

if(f>threshold)

return f;

if(node==Goal)

return FOUND;

integer min=MAX_INT;

foreach(tempnode in nextnodes(node))

```

{
integer temp=search(tempnode,g+cost(node,tempnode),threshold);
if(temp==FOUND)
return FOUND;
if(temp<min)
    threshold encountered
    min=temp;
}

return min; //return the minimum 'f' encountered greater than threshold
}
function nextnodes(node)
{
    return list of all possible next nodes from node;
}

```

Q:8 Explain resolution principle.

Ans: In mathematical logic and automated theorem proving, **resolution** is a rule of inference leading to a refutation theorem-proving technique for sentences in propositional logic and first-order logic. In other words, iteratively applying the resolution rule in a suitable way allows for telling whether a propositional formula is satisfiable and for proving that a first-order formula is unsatisfiable. Attempting to prove a satisfiable first-order formula as unsatisfiable may result in a nonterminating computation; this problem doesn't occur in propositional logic.

RESOLUTION IN PROPOSITIONAL LOGIC:

Resolution rule:

The **resolution rule** in propositional logic is a single valid inference rule that produces a new clause implied by two **clauses** containing complementary literals. A **literal** is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other. The resulting clause contains all the literals that do not have complements. Formally:

$$\frac{a_1 \vee a_2 \vee \dots \vee c, \quad b_1 \vee b_2 \vee \dots \vee \neg c}{a_1 \vee a_2 \vee \dots \vee b_1 \vee b_2 \vee \dots}$$

where,

All a_i , b_i and c are literals,

the dividing line stands for "entails".

The above may also be written as:

$$\frac{(\neg a_1 \wedge \neg a_2 \wedge \dots) \rightarrow c, \quad c \rightarrow (b_1 \vee b_2 \vee \dots)}{(\neg a_1 \wedge \neg a_2 \wedge \dots) \rightarrow (b_1 \vee b_2 \vee \dots)}$$

The clause produced by the resolution rule is called the *resolvent* of the two input clauses. It is the principle of consensus applied to clauses rather than terms. When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology.

Modus ponens can be seen as a special case of resolution (of a one-literal clause and a two-literal clause).

$$\frac{p \rightarrow q, \quad p}{q}$$

can be considered equal to

$$\frac{\neg p \vee q, \quad p}{q}$$

Resolution Technique:

When coupled with a complete search algorithm, the resolution rule yields a sound and complete algorithm for deciding the *satisfiability* of a propositional formula, and, by extension, the validity of a sentence under a set of axioms.

This resolution technique uses proof by contradiction and is based on the fact that any sentence in propositional logic can be transformed into an equivalent sentence in conjunctive normal form. The steps are as follows.

- All sentences in the knowledge base and the *negation* of the sentence to be proved (the *conjecture*) are conjunctively connected.
- The resulting sentence is transformed into a conjunctive normal form with the conjuncts viewed as elements in a set, S , of clauses.

For example $(A_1 \vee A_2) \wedge (B_1 \vee B_2 \vee B_3) \wedge (C_1)$ gives rise to the set
 $S = \{A_1 \vee A_2, B_1 \vee B_2 \vee B_3, C_1\}$

- The resolution rule is applied to all possible pairs of clauses that contain complementary literals. After each application of the resolution rule, the resulting sentence is simplified by removing repeated literals. If the sentence contains complementary literals, it is discarded (as a tautology). If not, and if it is not yet present in the clause set S , it is added to S , and is considered for further resolution inferences.
- If after applying a resolution rule the *empty clause* is derived, the original formula is unsatisfiable (or *contradictory*), and hence it can be concluded that the initial conjecture follows from the axioms.

- If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, the conjecture is not a theorem of the original knowledge base.

Q:9 Explain various operators over the fuzzy set.

Ans:

Given 'X' to be universe of discourse, A and B are two fuzzy sets with membership function $\mu_A(x)$ and $\mu_B(x)$ then,

1. Union:


The union of two fuzzy sets A and B is a new fuzzy set $A \cup B$ also on 'X' with membership function defined as follow:

$$\mu_{(A \cup B)} = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

2. Intersection:

Intersection of fuzzy sets A & B, is a new fuzzy set $A \cap B$ also on 'X' whose membership function is defined by

$$\mu_{(A \cap B)} = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$



 minimum operator

3. Compliment:

Compliment of a fuzzy set A is \bar{A} with membership function

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

4. Product of Two Fuzzy Sets:

The product of two fuzzy sets A & B is a new fuzzy set $A.B$ with membership function:

$$\mu_{A.B}(x) = \mu_A(x) \cdot \mu_B(x)$$

5. Equality:

Two fuzzy sets A and B are said to be equal i.e, $A = B$ if and only if $\mu_A(x) = \mu_B(x)$ Which means their membership values must be equal.

6. Product of Fuzzy Sets with a Crisp Number:

Multiplying a fuzzy set A by a crisp number 'n' results in a new fuzzy set $n.A$, whose membership function is

$$\mu_{n.A}(x) = n \cdot \mu_A(x)$$

7. Power of a Fuzzy Set:

The alpha power of a fuzzy set A is a new fuzzy set A^α whose membership function is: that is, individual memberships power of α

$$\mu_{A^\alpha}(x) = [\mu_A(x)]^\alpha$$

8. Difference of Fuzzy Sets

The differences of two fuzzy sets A and B is a new fuzzy set $A-B$ which is defined as

$$A - B = (A \cap \overline{B})$$

9. Disjunctive Sum of A & B

It is the new fuzzy set defined as follow:

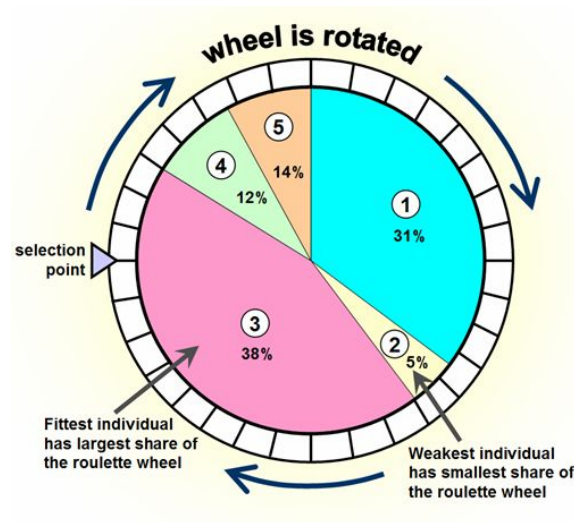
$$A \oplus B = (\overline{A} \cap B) \cup (A \cap \overline{B})$$

Q:10 Types of Selection.

Ans.

Roulette wheel selection

The basic part of the selection process is to stochastically select from one generation to create the basis of the next generation. The requirement is that the fittest individuals have a greater chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation. Weaker individuals are not without a chance. In nature such individuals may have genetic coding that may prove useful to future generations.



Tournament Selection

Tournament Selection is a Selection Strategy used for selecting the fittest candidates from the current generation in a Genetic Algorithm. These selected candidates are then passed on to the next generation. In a K-way tournament selection, we select k-individuals and run a tournament among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation. In this way many such tournaments take place and we have our final selection of candidates who move on to the next generation. It also has a parameter called the selection pressure which is a probabilistic measure of a candidate's likelihood of participation in a tournament. If the tournament size is larger, weak candidates have a smaller chance of getting selected as it has to compete with a stronger candidate.

Algorithm

1. Select k individuals from the population and perform a tournament amongst them
2. Select the best individual from the k individuals
3. Repeat process 1 and 2 until you have the desired amount of population

Q:11 Types of crossover.

Ans.

Crossover with Reduced Surrogate

This technique can limit the crossover points that can be chosen to positions where the values in the two parents differ. As a result, positions where the values are swapped will always cause a change, and hence produce new individuals whenever possible. This technique is performed in conjunction with the others

Arithmetic Crossover

Arithmetic crossover is used in case of real-value encoding. Arithmetic crossover operator linearly combines the two parent chromosomes [17]. Two chromosomes are particular randomly for crossover and create two offspring's which are linear mixture of their parents.

Partially mapped Crossover

Partially Matched or Mapped Crossover (PMX) is the most frequently used crossover operator. It was proposed by Goldberg and Lingle [19] for Travelling Salesman Problem. In Partially Matched Crossover, two chromosomes are associated and two crossover sites are chosen arbitrarily. The fraction of chromosomes between the two crossover points gives a corresponding selection that undergoes the crossover process through position-by-position exchange operations [2, 17]. PMX tends to respect the absolute positions.

Shuffle Crossover

Shuffle Crossover helps in creation of offspring which have independent of crossover point in their parents. It uses the same 1-Point Crossover technique in addition to shuffle. Shuffle Crossover selects the two parents for crossover. It firstly randomly shuffles the genes in the both parents but in the same way. Then it applies the 1-Point crossover technique by randomly selecting a point as crossover point and then combines both parents to create two offspring. After performing 1-point crossover the genes in offspring are then unshuffled in same way as they have been shuffled.

Precedence Preservative Crossover (PPX)

Precedence Preservative Crossover was independently developed for vehicle routing problems by Blanton and Wainwright (1993) and for scheduling problems by Bierwirth et al. (1996). The operator passes on precedence relations of operations given in two parental permutations to one offspring at the same rate, while no new precedence relations are introduced. PPX is illustrated in below, for a problem consisting of six operations A–F. The operator works as follows:

- A vector of length Sigma, sub $i=1$ to m , representing the number of operations involved in the problem, is randomly filled with elements of the set $\{1, 2\}$.
- This vector defines the order in which the operations are successively drawn from parent 1 and parent 2.

- We can also consider the parent and offspring permutations as lists, for which the operations 'append' and 'delete' are defined.
- First we start by initializing an empty offspring.
- The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.
- After an operation is selected it is deleted in both parents. 2
- Finally the selected operation is appended to the offspring.
- This step is repeated until both parents are empty and the offspring contains all operations involved.
- Note that PPX does not work in a uniform-crossover manner due to the 'deletion append' scheme used.

Example is shown in Figure below

Parent permutation 1	A	B	C	D	E	F
Parent permutation 2	C	A	B	F	D	E
Select parent no. (1/2)	1	2	1	1	2	2
Offspring permutation	A	C	B	D	F	E

Reference:

1. <http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php>
2. <https://www.geeksforgeeks.org/tournament-selection-ga/>
3. http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf
4. <http://ijcsit.com/docs/Volume%205/vol5issue06/ijcsit2014050673.pdf>
5. http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf
6. http://repository.uobabylon.edu.iq/mirror/resources/paper_2_2712_124.pdf
7. http://www.uobabylon.edu.iq/uobcoleges/ad_downloads/4_10592_215.pdf

backpropagation

April 15, 2019

1 Backpropagation

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for “the backward propagation of errors,” since an error is computed at the output and distributed backwards throughout the network’s layers. It is commonly used to train deep neural networks.

```
In [1]: from math import exp
        from random import seed
        from random import random
```

```
In [2]: # Initialize a network
        def initialize_network(n_inputs, n_hidden, n_outputs):
            network = list()
            hidden_layer = [{'weights': [random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
            network.append(hidden_layer)
            output_layer = [{'weights': [random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
            network.append(output_layer)
            return network
```

```
In [3]: # Calculate neuron activation for an input
        def activate(weights, inputs):
            activation = weights[-1]
            for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
            return activation
```

```
In [4]: # Transfer neuron activation
        def transfer(activation):
            return 1.0 / (1.0 + exp(-activation))
```

```
In [5]: # Forward propagate input to a network output
        def forward_propagate(network, row):
            inputs = row
            for layer in network:
                new_inputs = []
                for neuron in layer:
```

```

        activation = activate(neuron['weights'], inputs)
        neuron['output'] = transfer(activation)
        new_inputs.append(neuron['output'])
    inputs = new_inputs
    return inputs

In [6]: # Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

In [7]: # Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

In [8]: # Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

In [9]: # Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])

```

```

        backward_propagate_error(network, expected)
        update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

In [10]: # Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
            [1.465489372,2.362125076,0],
            [3.396561688,4.400293529,0],
            [1.38807019,1.850220317,0],
            [3.06407232,3.005305973,0],
            [7.627531214,2.759262235,1],
            [5.332441248,2.088626775,1],
            [6.922596716,1.77106367,1],
            [8.675418651,-0.242068655,1],
            [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)

>epoch=0, lrate=0.500, error=6.365
>epoch=1, lrate=0.500, error=5.557
>epoch=2, lrate=0.500, error=5.291
>epoch=3, lrate=0.500, error=5.262
>epoch=4, lrate=0.500, error=5.217
>epoch=5, lrate=0.500, error=4.899
>epoch=6, lrate=0.500, error=4.419
>epoch=7, lrate=0.500, error=3.900
>epoch=8, lrate=0.500, error=3.461
>epoch=9, lrate=0.500, error=3.087
>epoch=10, lrate=0.500, error=2.758
>epoch=11, lrate=0.500, error=2.468
>epoch=12, lrate=0.500, error=2.213
>epoch=13, lrate=0.500, error=1.989
>epoch=14, lrate=0.500, error=1.792
>epoch=15, lrate=0.500, error=1.621
>epoch=16, lrate=0.500, error=1.470
>epoch=17, lrate=0.500, error=1.339
>epoch=18, lrate=0.500, error=1.223
>epoch=19, lrate=0.500, error=1.122
[{'weights': [-0.9766426647918854, 1.0573043092399, 0.7999535671683315], 'output': 0.054299270}
[{'weights': [1.4965066037208181, 1.770264295168642, -1.28526000789383], 'output': 0.246982887}

```


1.1 References:

1. <https://en.wikipedia.org/wiki/Backpropagation>
2. <https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>
3. <https://www.youtube.com/watch?v=aircAruvnKk&list=PLLMP7TazTxHrgVk7w1EKpLBIDoC50QrPS&y>
4. <http://neuralnetworksanddeeplearning.com/chap2.html>

hopfield

April 15, 2019

1 Hopfield Neural Network

A Hopfield network is a form of recurrent artificial neural network popularized by John Hopfield in 1982, but described earlier by Little in 1974. Hopfield nets serve as content-addressable (“associative”) memory systems with binary threshold nodes. They are guaranteed to converge to a local minimum and, therefore, may converge to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum). Hopfield networks also provide a model for understanding human memory.

```
In [1]: # importing modules
```

```
import numpy as np
from neupy import algorithms
from neupy import plots
import matplotlib.pyplot as plt
```

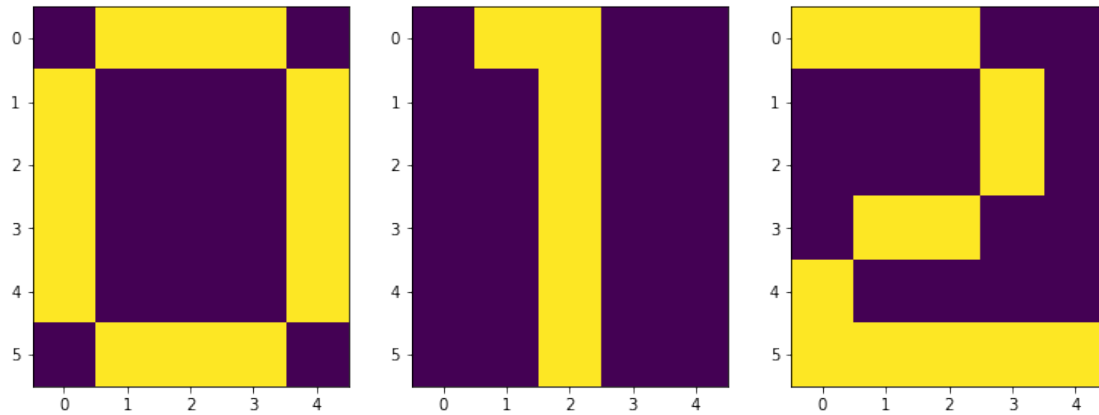
```
In [2]: # Training Data
```

```
zero = np.matrix([ 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
one = np.matrix([0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
two = np.matrix([1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
```

```
data = np.concatenate([zero, one, two], axis=0)
```

```
plt.figure(figsize=[12,12])
plt.subplot(131)
plt.imshow(zero.reshape(6,5))
plt.subplot(132)
plt.imshow(one.reshape(6,5))
plt.subplot(133)
plt.imshow(two.reshape(6,5))
```

```
Out[2]: <matplotlib.image.AxesImage at 0x7fe8fafc0278>
```



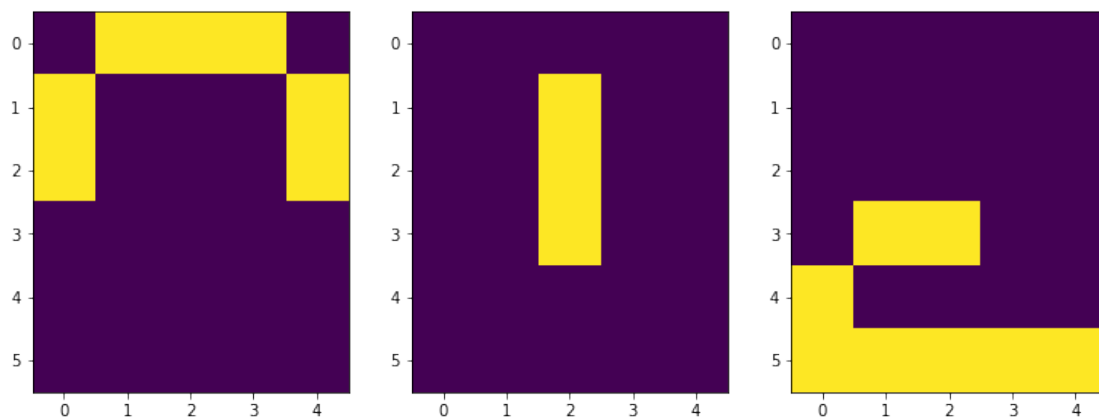
In [3]: *# model training*

```
hop = algorithms.DiscreteHopfieldNetwork(mode='sync')
hop.train(data)
```

In [4]: *# Testing Data*

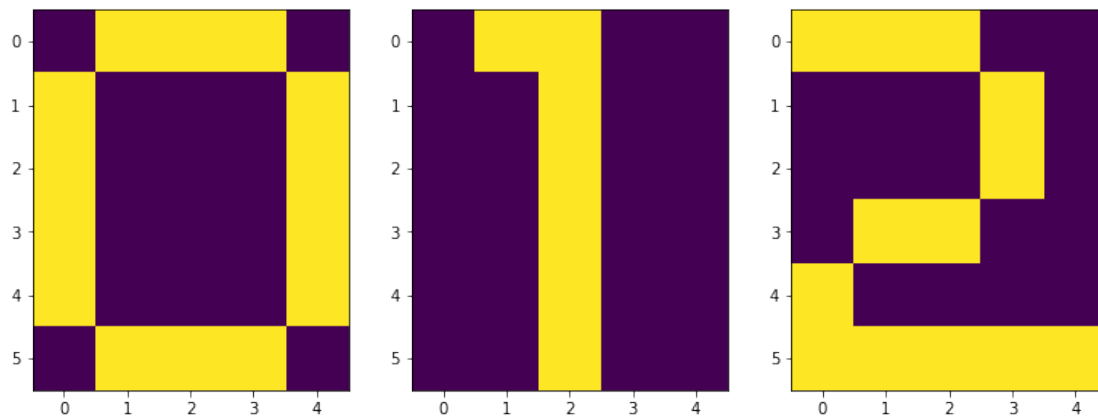
```
half_zero = np.matrix([0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
half_two = np.matrix([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0])
half_one = np.matrix([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0])
plt.figure(figsize=[12,12])
plt.subplot(131)
plt.imshow(half_zero.reshape(6,5))
plt.subplot(132)
plt.imshow(half_one.reshape(6,5))
plt.subplot(133)
plt.imshow(half_two.reshape(6,5))
```

Out[4]: <matplotlib.image.AxesImage at 0x7fe8faf02198>



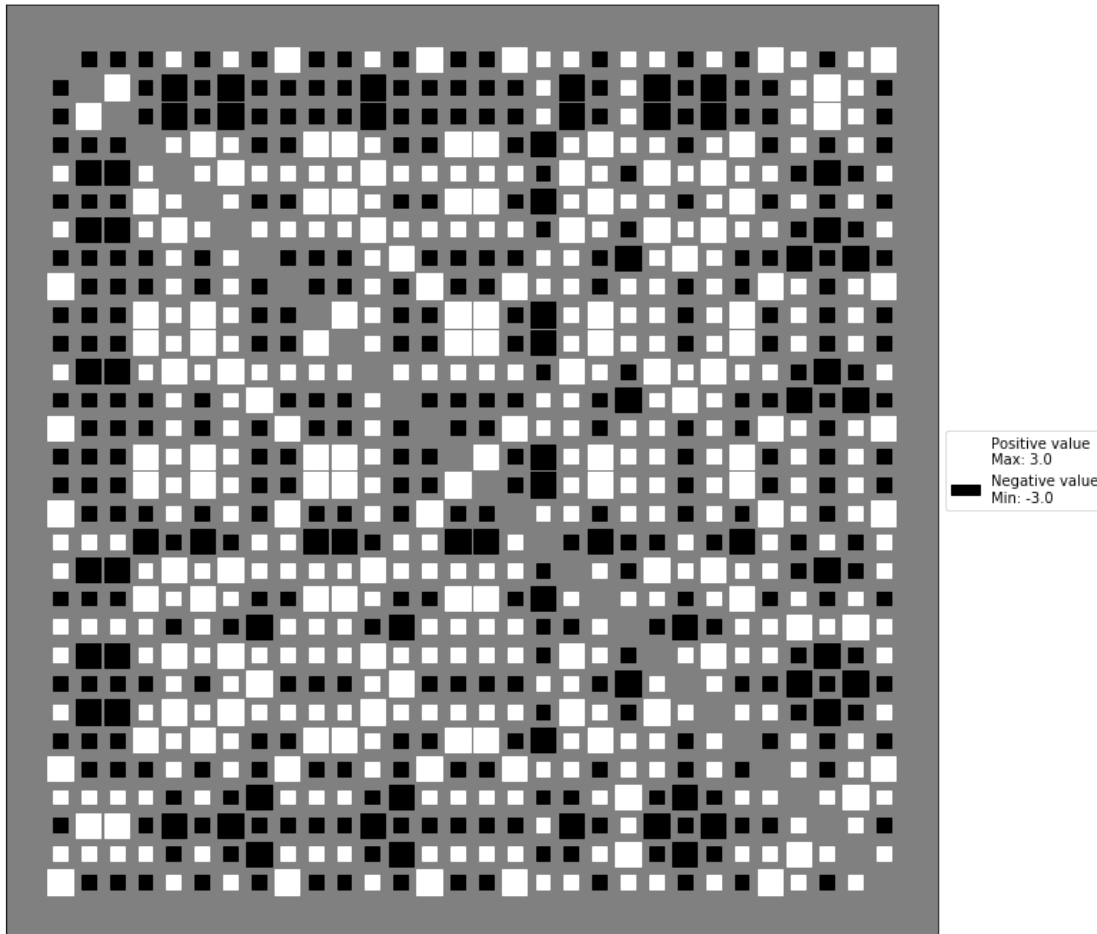
```
In [5]: # pattern reconstruction
recon_zero = hop.predict(half_zero)
recon_one = hop.predict(half_one)
recon_two = hop.predict(half_two)
plt.figure(figsize=[12,12])
plt.subplot(131)
plt.imshow(recon_zero.reshape(6,5))
plt.subplot(132)
plt.imshow(recon_one.reshape(6,5))
plt.subplot(133)
plt.imshow(recon_two.reshape(6,5))
```

Out[5]: <matplotlib.image.AxesImage at 0x7fe8fae73be0>



```
In [6]: plt.figure(figsize=(14, 12))
plots.hinton(hop.weight)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8fadc2208>



1.1 References

1. http://neupy.com/2015/09/20/discrete_hopfield_network.html#discrete-hopfield-network
2. https://en.wikipedia.org/wiki/Hopfield_network

sk_kMeans

April 15, 2019

1 k-means using sklearn

1.0.1 Simple algorithm for K-means clustering

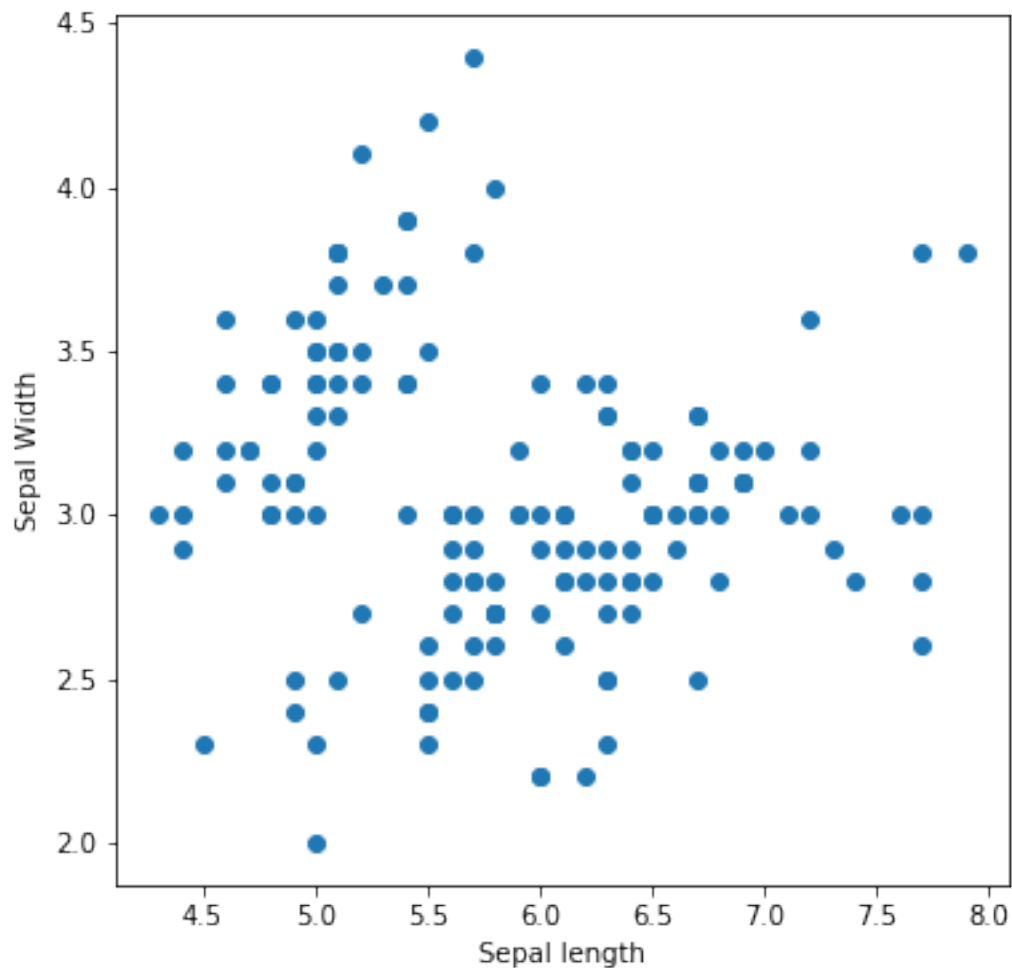
1. Find the Euclidean distance between each data instance and centroids of all the clusters
2. Assign the data instances to the cluster of the centroid with nearest distance
3. Calculate new centroid values based on the mean values of the coordinates of all the data instances from the corresponding cluster.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

```
In [2]: dataset = datasets.load_iris()
dataset.feature_names
```

```
Out[2]: ['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)']
```

```
In [3]: # Feature selection
X = dataset.data[:, np.array([True, True, False, True])]
plt.figure(figsize=(6, 6))
plt.scatter( X[:, 0], X[:, 1])
plt.xlabel('Sepal length')
plt.ylabel('Sepal Width')
plt.show()
```



```
In [4]: # Model
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

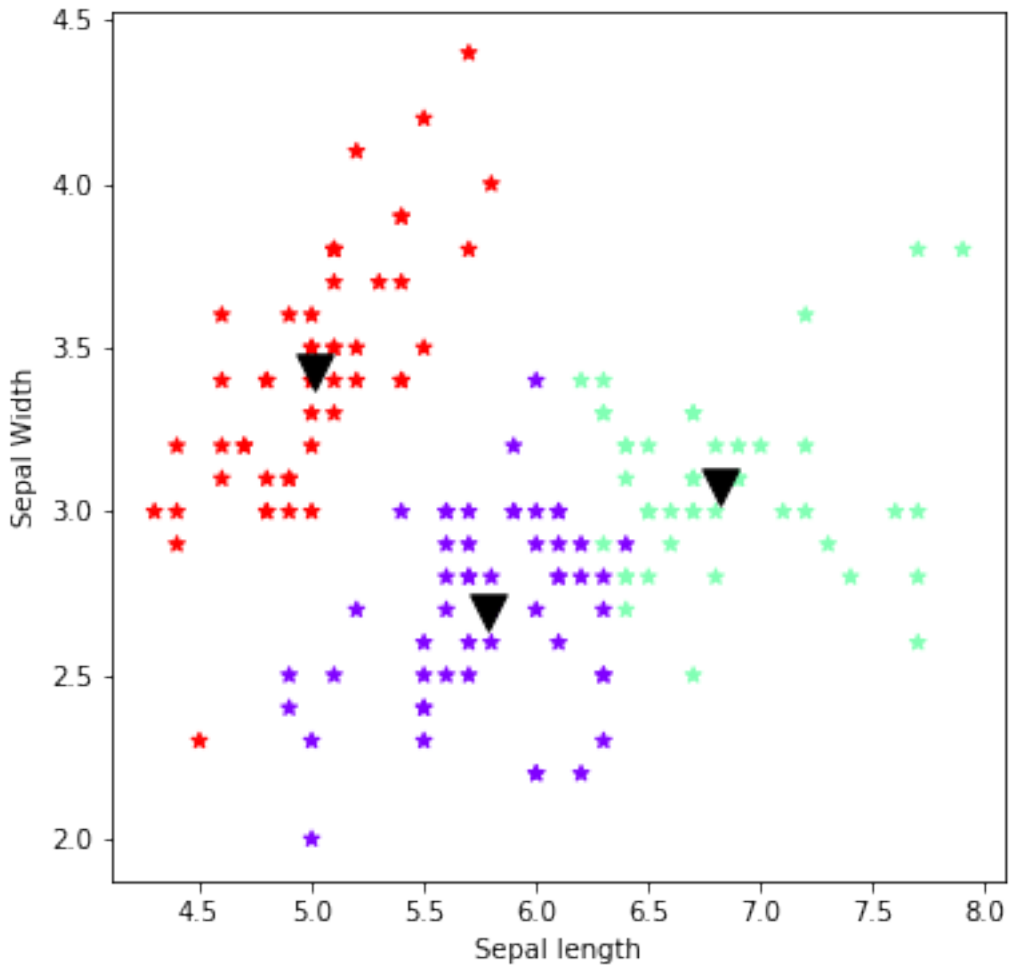
```
Out[4]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
               n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
               random_state=None, tol=0.0001, verbose=0)
```

```
In [5]: print(kmeans.cluster_centers_)
```

```
[[5.78518519 2.6962963 1.43148148]
 [6.82173913 3.07826087 1.96304348]
 [5.006      3.428      0.246      ]]
```

```
In [6]: markers = ["*", "v", "s"]
plt.figure(figsize=(6, 6))
```

```
plt.scatter( X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow', marker="*")
plt.scatter(kmeans.cluster_centers_[0,0] ,kmeans.cluster_centers_[0,1], color='black',
plt.xlabel('Sepal length')
plt.ylabel('Sepal Width')
plt.show()
```



1.1 References:

1. <https://stackabuse.com/k-means-clustering-with-scikit-learn/>
2. <https://stackoverflow.com/questions/28296670/remove-a-specific-feature-in-scikit-learn>

sk_naiveBayes

April 15, 2019

1 Naive Bayes

Naive Bayes uses Bayes Theorem to model the conditional relationship of each attribute to the class variable.

```
In [1]: from sklearn import datasets
        from sklearn import metrics
        from sklearn.naive_bayes import GaussianNB
```

2 Iris flowe Dataset

```
In [2]: dataset = datasets.load_iris()
        dataset.feature_names
```

```
Out[2]: ['sepal length (cm)',
         'sepal width (cm)',
         'petal length (cm)',
         'petal width (cm)']
```

3 Model

```
In [3]: model = GaussianNB()
        model.fit(dataset.data, dataset.target)
```

```
Out[3]: GaussianNB(priors=None, var_smoothing=1e-09)
```

4 Prediction/Classification

```
In [4]: expected = dataset.target
        predicted = model.predict(dataset.data)
        print(metrics.classification_report(expected, predicted))
        print(metrics.confusion_matrix(expected, predicted))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50

1	0.94	0.94	0.94	50
2	0.94	0.94	0.94	50
micro avg	0.96	0.96	0.96	150
macro avg	0.96	0.96	0.96	150
weighted avg	0.96	0.96	0.96	150

```

[[50 0 0]
 [ 0 47 3]
 [ 0 3 47]]

```

4.1 References

1. <https://machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/>

GIBBS SAMPLING

In [statistics](#), **Gibbs sampling** or a **Gibbs sampler** is a [Markov chain Monte Carlo](#) (MCMC) [algorithm](#) for obtaining a sequence of observations which are approximated from a specified [multivariate probability distribution](#), when direct sampling is difficult. This sequence can be used to approximate the joint distribution (e.g., to generate a histogram of the distribution); to approximate the [marginal distribution](#) of one of the variables, or some subset of the variables (for example, the unknown [parameters](#) or [latent variables](#)); or to compute an [integral](#) (such as the [expected value](#) of one of the variables). Typically, some of the variables correspond to observations whose values are known, and hence do not need to be sampled.

Gibbs sampling is commonly used as a means of [statistical inference](#), especially [Bayesian inference](#). It is a [randomized algorithm](#) (i.e. an algorithm that makes use of [random numbers](#)), and is an alternative to [deterministic algorithms](#) for statistical inference such as the [expectation-maximization algorithm](#) (EM).

As with other MCMC algorithms, Gibbs sampling generates a [Markov chain](#) of samples, each of which is [correlated](#) with nearby samples. As a result, care must be taken if independent samples are desired. Generally, samples from the beginning of the chain (the *burn-in period*) may not accurately represent the desired distribution and are usually discarded. It has been shown, however, that using a longer chain instead (e.g. a chain that is n times as long as the initially considered chain using a thinning factor of n) leads to better estimates of the true posterior. Thus, *thinning* should only be applied when time or computer memory are restricted.

GIBBS ALGORITHM

Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply. The expense is due to the fact that it computes the posterior probability for every hypothesis in H and then combines the predictions of each hypothesis to classify each new instance. An alternative, less optimal method is the Gibbs algorithm (see Oppen and Haussler 1991), defined as follows:

1. Choose a hypothesis h from H at random, according to the posterior probability distribution over H .
2. Use h to predict the classification of the next instance x .

Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution. Surprisingly, it can be shown that under certain conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier (Haussler et al. 1994). More precisely, the expected value is taken over target concepts drawn at random according to the prior probability distribution assumed by the learner. Under this condition, the expected value of the error of the Gibbs algorithm is at worst twice the expected value of the error of the Bayes optimal classifier. This result has an interesting implication for the concept learning problem described earlier. In particular, it implies that if

the learner assumes a uniform prior over H , and if target concepts are in fact drawn from such a distribution when presented to the learner, then classifying the next instance according to a hypothesis drawn at random from the current version space (according to a uniform distribution), will have expected error at most twice that of the Bayes optimal classifier. Again, we have an example where a Bayesian analysis of a non-Bayesian algorithm yields insight into the performance of that algorithm.

References:

1. <http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf>
2. <https://www.mit.edu/~ilkery/papers/GibbsSampling.pdf>
3. <https://stats.stackexchange.com/questions/325696/explanation-regarding-gibbs-sampling>
4. <https://kieranrcampbell.github.io/blog/2016/05/15/gibbs-sampling-bayesian-linear-regression.html>
5. <https://stats.stackexchange.com/questions/325696/explanation-regarding-gibbs-sampling>

sk_randomForest

April 15, 2019

1 Ransom Forest Algorithm

Random forest algorithm is an ensemble classification algorithm. Ensemble classifier means a group of classifiers. Instead of using only one classifier to predict the target, In ensemble, we use multiple classifiers to predict the target.

In case, of random forest, these ensemble classifiers are the randomly created decision trees. Each decision tree is a single classifier and the target prediction is based on the majority voting method.

The majority voting concept is same as the political votings. Each person votes per one political party out all the political parties participating in elections. In the same way, every classifier will votes to one target class out of all the target classes.

To declare the election results. The votes will calculate and the party which got the most number of votes treated as the election winner. In the same way, the target class which got the most number of votes considered as the final predicted target class.

```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score, confusion_matrix
        from sklearn import datasets
```

```
In [2]: dataset = datasets.load_breast_cancer()
```

```
In [3]: cancer = pd.DataFrame(dataset.data, columns = dataset.feature_names)
        cancer.head()
```

```
Out[3]:
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	
3	0.28390	0.2414	0.10520	0.2597	

4	0.13280	0.1980	0.10430	0.1809
---	---------	--------	---------	--------

	mean fractal dimension	...	worst radius	\
0	0.07871	...	25.38	
1	0.05667	...	24.99	
2	0.05999	...	23.57	
3	0.09744	...	14.91	
4	0.05883	...	22.54	

	worst texture	worst perimeter	worst area	worst smoothness	\
0	17.33	184.60	2019.0	0.1622	
1	23.41	158.80	1956.0	0.1238	
2	25.53	152.50	1709.0	0.1444	
3	26.50	98.87	567.7	0.2098	
4	16.67	152.20	1575.0	0.1374	

	worst compactness	worst concavity	worst concave points	worst symmetry	\
0	0.6656	0.7119	0.2654	0.4601	
1	0.1866	0.2416	0.1860	0.2750	
2	0.4245	0.4504	0.2430	0.3613	
3	0.8663	0.6869	0.2575	0.6638	
4	0.2050	0.4000	0.1625	0.2364	

	worst fractal dimension
0	0.11890
1	0.08902
2	0.08758
3	0.17300
4	0.07678

[5 rows x 30 columns]

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(cancer, dataset.target)
```

```
In [5]: # model
rndmfrst = RandomForestClassifier(n_estimators=1000)
rndmfrst.fit(X_train, y_train)
```

```
Out [5]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
                                oob_score=False, random_state=None, verbose=0,
                                warm_start=False)
```

```
In [6]: # Prediction
y_pred = rndmfrst.predict(X_test)
```

```

    for i in range(0, 5):
        print("Actual outcome :: {} and predicted outcome :: {}".format(y_test[i], y_pred[i]))

Actual outcome :: 1 and predicted outcome :: 1
Actual outcome :: 0 and predicted outcome :: 0
Actual outcome :: 0 and predicted outcome :: 0
Actual outcome :: 0 and predicted outcome :: 0
Actual outcome :: 0 and predicted outcome :: 1

In [7]: # Train and Test Accuracy
        print ("Train Accuracy :: ", accuracy_score(y_train, rndmfrst.predict(X_train)))
        print ("Test Accuracy  :: ", accuracy_score(y_test, y_pred))
        print ("Confusion matrix :: \n", confusion_matrix(y_test, y_pred))

Train Accuracy ::  1.0
Test Accuracy  ::  0.958041958041958
Confusion matrix ::
[[54  6]
 [ 0 83]]

```

1.1 References:

1. <http://dataaspirant.com/2017/06/26/random-forest-classifier-python-scikit-learn/>
2. https://en.wikipedia.org/wiki/Random_forest
3. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

sk_kNearestNeighbor

April 15, 2019

1 k-Nearest Neighbor

The k-Nearest Neighbor (kNN) method makes predictions by locating similar cases to a given data instance (using a similarity function) and returning the average or majority of the most similar data instances. The kNN algorithm can be used for classification or regression.

KNN falls in the supervised learning family of algorithms. Informally, this means that we are given a labelled dataset consisting of training observations (x,y) and would like to capture the relationship between x and y. More formally, our goal is to learn a function $h:XY$ so that given an unseen observation x, $h(x)$ can confidently predict the corresponding output y

```
In [1]: from sklearn import datasets
        from sklearn import metrics
        from sklearn.neighbors import KNeighborsClassifier
        import matplotlib.pyplot as plt
        import numpy as np
```

2 Iris flowers Dataset

```
In [2]: dataset = datasets.load_iris()
        dataset.feature_names
```

```
Out[2]: ['sepal length (cm)',
         'sepal width (cm)',
         'petal length (cm)',
         'petal width (cm)']
```

3 Model

```
In [3]: model = KNeighborsClassifier()
        model.fit(dataset.data, dataset.target)
```

```
Out[3]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                             weights='uniform')
```


4 Prediction/Classification

```
In [4]: expected = dataset.target
        predicted = model.predict(dataset.data)
        print("\nClassification Report : \n\n", metrics.classification_report(expected, predicted))
        print("\nConfusion Matrix: \n\n", metrics.confusion_matrix(expected, predicted))
```

Classification Report :

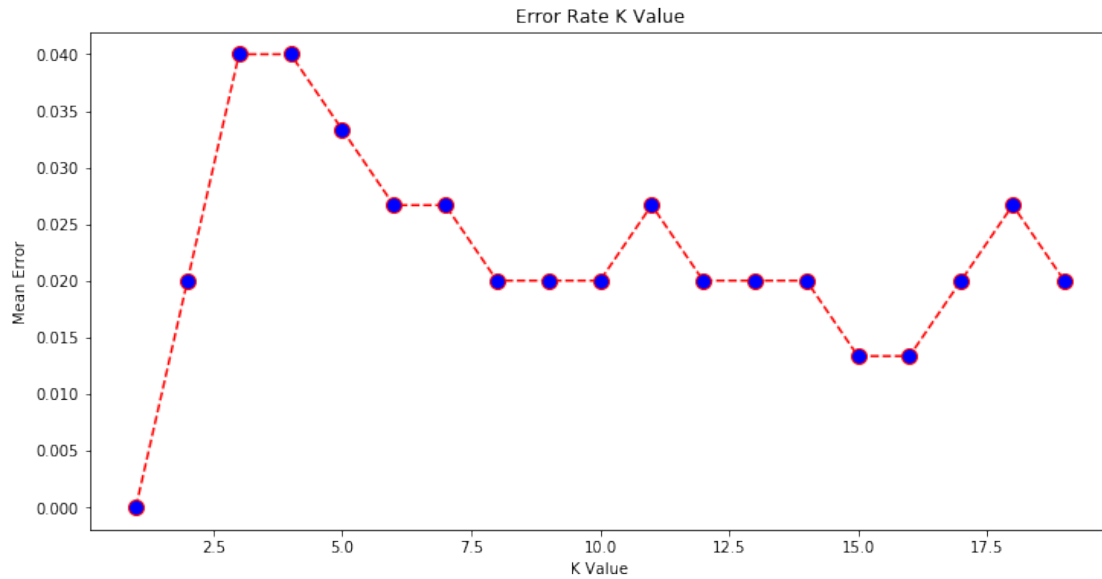
	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.96	0.94	0.95	50
2	0.94	0.96	0.95	50
micro avg	0.97	0.97	0.97	150
macro avg	0.97	0.97	0.97	150
weighted avg	0.97	0.97	0.97	150

Confusion Matrix:

```
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]
```

```
In [5]: # Calculating error for K values between 1 and n
        n = 20
        error = []
        for i in range(1, n):
            knn = KNeighborsClassifier(n_neighbors=i)
            knn.fit(dataset.data, dataset.target)
            pred_i = knn.predict(dataset.data)
            error.append(np.mean(pred_i != dataset.target))

        plt.figure(figsize=(12, 6))
        plt.plot(range(1, n), error, color='red', linestyle='dashed', marker='o',
                  markerfacecolor='blue', markersize=10)
        plt.title('Error Rate K Value')
        plt.xlabel('K Value')
        plt.ylabel('Mean Error')
        plt.show()
```



4.1 References

1. <https://machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/>
2. <https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/>
3. <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>

sk_linearRegression

April 15, 2019

1 Linear regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

```
In [1]: # linear regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn import metrics

In [2]: dataset = datasets.load_diabetes()
dataset.feature_names

Out[2]: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']

In [3]: # feature selection
bmi = dataset.data[:, np.newaxis ,2]
# train test split
X_train, X_test, y_train, y_test = train_test_split(bmi, dataset.target, test_size=0.2)

In [4]: # model
regressor = linear_model.LinearRegression()
regressor.fit(X_train.reshape(-1, 1) , y_train.reshape(-1, 1) )

Out[4]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)

In [5]: print(regressor.intercept_)
print(regressor.coef_)

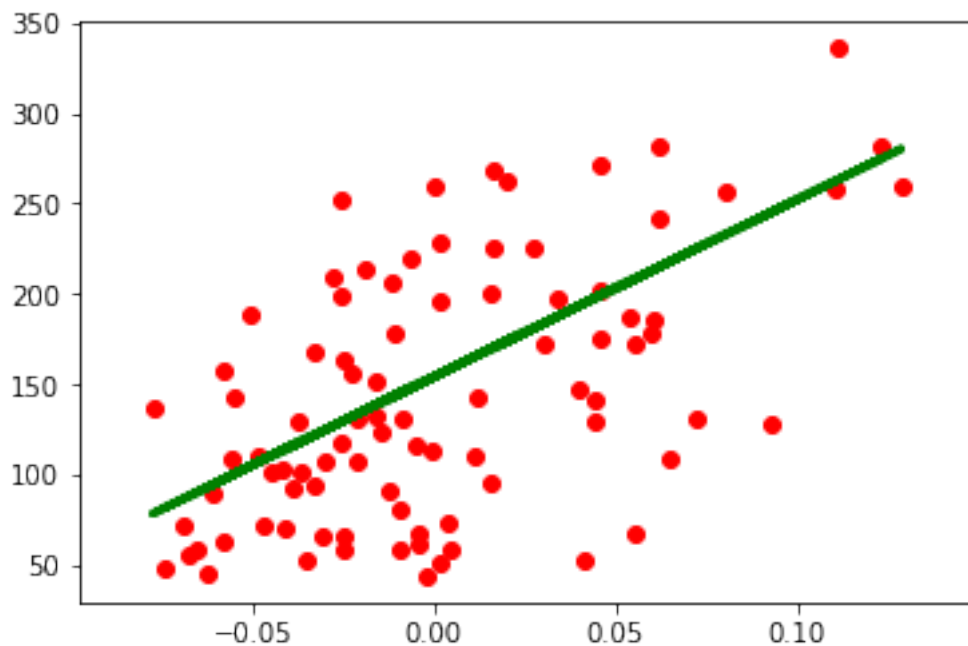
[154.27729208]
[[979.8299551]]
```

```
In [6]: # prediction for test data
        y_pred = regressor.predict(X_test)

In [7]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
        print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
        print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
        print('Variance score: %.2f' % metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 49.4200929535062
Mean Squared Error: 3612.4239077907882
Root Mean Squared Error: 60.10344339379224
Variance score: 0.27
```

```
In [8]: plt.scatter(X_test, y_test, color='red')
        plt.plot(X_test, y_pred, color='green', linewidth=3)
        plt.show()
```



1.1 References:

1. https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html
2. <https://stackabuse.com/linear-regression-in-python-with-scikit-learn/>
3. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
4. https://en.wikipedia.org/wiki/Linear_regression

sk_logisticRegression

April 15, 2019

1 Logistic Regression

Logistic regression fits a logistic model to data and makes predictions about the probability of an event (between 0 and 1).

```
In [1]: from sklearn import datasets
        from sklearn import metrics
        from sklearn.linear_model import LogisticRegression
```

2 Iris flowers Dataset

```
In [2]: dataset = datasets.load_iris()
        dataset.feature_names
```

```
Out[2]: ['sepal length (cm)',
         'sepal width (cm)',
         'petal length (cm)',
         'petal width (cm)']
```

3 MODEL

```
In [3]: # creating a model
        model = LogisticRegression(random_state=0, solver='lbfgs', multi_class='auto', max_iter=1000)
        model.fit(dataset.data, dataset.target)
```

```
Out[3]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                           intercept_scaling=1, max_iter=1000, multi_class='auto',
                           n_jobs=None, penalty='l2', random_state=0, solver='lbfgs',
                           tol=0.0001, verbose=0, warm_start=False)
```

4 lbfgs solver

Limited-memory BFGS is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm using a limited amount of computer memory. It is a popular algorithm for parameter estimation in machine learning.

5 Prediction/Classification

```
In [4]: expected = dataset.target
        predicted = model.predict(dataset.data)
        print(metrics.classification_report(expected, predicted))
        print(metrics.confusion_matrix(expected, predicted))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.98	0.94	0.96	50
2	0.94	0.98	0.96	50
micro avg	0.97	0.97	0.97	150
macro avg	0.97	0.97	0.97	150
weighted avg	0.97	0.97	0.97	150


```
[[50  0  0]
 [ 0 47  3]
 [ 0  1 49]]
```

6 Classification Report variables

6.1 Precision

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. ### $\text{Precision} = \frac{\text{truePositive}}{(\text{truePositive} + \text{falsePositive})}$

6.2 Recall

The recall is intuitively the ability of the classifier to find all the positive samples. ### $\text{Recall} = \frac{\text{truePositive}}{(\text{truePositive} + \text{falseNegative})}$

6.3 F1-score

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. ### $\text{F1} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$

6.4 Summary

The f1-score gives you the harmonic mean of precision and recall. The scores corresponding to every class will tell you the accuracy of the classifier in classifying the data points in that particular class compared to all other classes. The support is the number of samples of the true response that lie in that class.

6.5 References

1. <https://machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/>
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression
3. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support
4. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score
5. https://en.wikipedia.org/wiki/Limited-memory_BFGS

sk_multipleRegression

April 15, 2019

1 Multiple Regression

Almost all real world problems that you are going to encounter will have more than two variables. Linear regression involving multiple variables is called “multiple linear regression”. The steps to perform multiple linear regression are almost similar to that of simple linear regression. The difference lies in the evaluation. You can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other.

```
In [1]: # Multiple linear regression
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split

In [2]: dataset = datasets.load_diabetes()
dataset.feature_names

Out[2]: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']

In [3]: # train test split
X_train, X_test, y_train, y_test = train_test_split(dataset.data, dataset.target, test_size=0.2)

In [4]: regressor = LinearRegression()
regressor.fit(X_train, y_train)

Out[4]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)

In [5]: print(">", regressor.intercept_)
print(">", regressor.coef_)

> 153.96242985120966
> [-55.62047247 -259.84522708  540.30079099  344.62715568 -971.71436208
  612.43396894  155.15860756  190.63482994  830.76852218  73.86967485]

In [6]: # prediction for test data
y_pred = regressor.predict(X_test)
```



```
In [7]: # Error report
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('Variance score: %.2f' % metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 42.097780223111066
Mean Squared Error: 2966.0402992842164
Root Mean Squared Error: 54.461365198498434
Variance score: 0.40
```

1.1 References:

1. <https://stackabuse.com/linear-regression-in-python-with-scikit-learn/>

sk_polynomialRegression

April 15, 2019

1 Polynomial Regression

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an n th degree polynomial in x .

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
```

```
In [2]: # dataset
boston = datasets.load_boston()
print(boston.data.shape, boston.target.shape)
print(boston.feature_names)
```

(506, 13) (506,)

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```
In [3]: # using pandas for data handling
data = pd.DataFrame(boston.data, columns=boston.feature_names)
data = pd.concat([data, pd.Series(boston.target, name='MEDV')], axis=1)
data.head()
```

```
Out[3]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

	PTRATIO	B	LSTAT	MEDV
0	15.3	396.90	4.98	24.0
1	17.8	396.90	9.14	21.6
2	17.8	392.83	4.03	34.7
3	18.7	394.63	2.94	33.4
4	18.7	396.90	5.33	36.2

```
In [4]: # Feature Selection
```

```
X = data[['LSTAT']]
```

```
y = data[['MEDV']]
```

```
In [5]: # train test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state=42)
```

```
In [6]: # Polynomial Regression nth order
```

```
plt.figure(figsize=(8, 4))
```

```
plt.scatter(X_test, y_test, alpha=0.3)
```

```
for degree in range(1, 4):
```

```
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
```

```
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
```

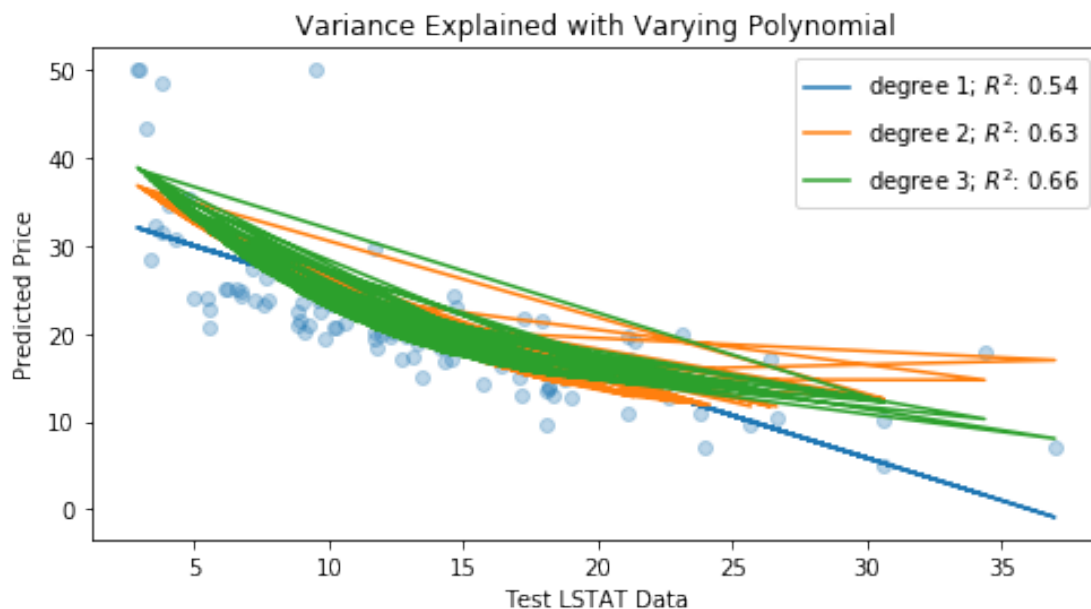
```
    plt.plot(X_test, y_pred, label="degree %d" % degree+';  $R^2$ : %.2f' % model.score(X_test, y_test))
```

```
    plt.legend(loc='upper right')
```

```
    plt.xlabel("Test LSTAT Data")
```

```
    plt.ylabel("Predicted Price")
```

```
    plt.title("Variance Explained with Varying Polynomial")
```



```

In [7]: # Polynomial interpolation
def f(x):
    """ function to approximate by polynomial interpolation """
    return x * np.sin(x)

# generate points used to plot
x_plot = np.linspace(0, 10, 100)

# generate points and keep a subset of them
x = np.linspace(0, 10, 100)
rng = np.random.RandomState(0)
rng.shuffle(x)
x = np.sort(x[:20])
y = f(x)

# create matrix versions of these arrays
X = x[:, np.newaxis]
X_plot = x_plot[:, np.newaxis]

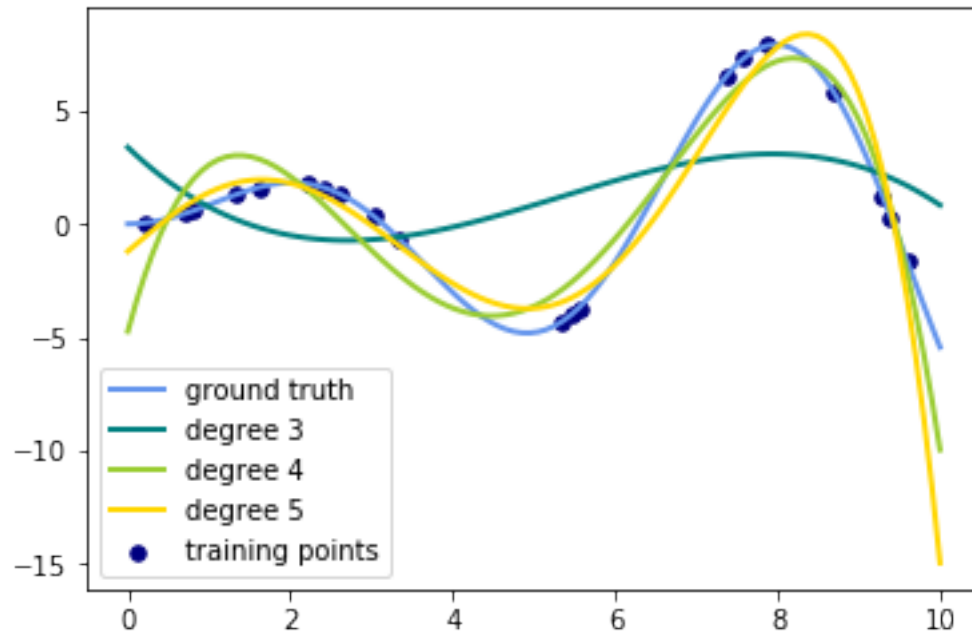
colors = ['teal', 'yellowgreen', 'gold']
lw = 2
plt.plot(x_plot, f(x_plot), color='cornflowerblue', linewidth=lw,
         label="ground truth")
plt.scatter(x, y, color='navy', s=30, marker='o', label="training points")

for count, degree in enumerate([3, 4, 5]):
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(X, y)
    y_plot = model.predict(X_plot)
    plt.plot(x_plot, y_plot, color=colors[count], linewidth=lw,
            label="degree %d" % degree)

plt.legend(loc='lower left')

plt.show()

```



1.1 References:

1. <https://acadgild.com/blog/polynomial-regression-understand-power-of-polynomials>
2. https://en.wikipedia.org/wiki/Polynomial_regression
3. https://scikit-learn.org/stable/auto_examples/linear_model/plot_polynomial_interpolation.html

sk_classification®ressionTree

April 15, 2019

1 Classification and Regression Trees

Classification and Regression Trees (CART) are constructed from a dataset by making splits that best separate the data for the classes or predictions being made. The CART algorithm can be used for classification or regression.

```
In [1]: from sklearn import datasets
        from sklearn import metrics
        from sklearn.tree import DecisionTreeClassifier
```

2 Iris flowers Dataset

```
In [2]: dataset = datasets.load_iris()
```

3 Model

```
In [3]: model = DecisionTreeClassifier()
        model.fit(dataset.data, dataset.target)
```

```
Out[3]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                                splitter='best')
```

4 Prediction/Classification

```
In [4]: expected = dataset.target
        predicted = model.predict(dataset.data)
        print(metrics.classification_report(expected, predicted))
        print(metrics.confusion_matrix(expected, predicted))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50

1	1.00	1.00	1.00	50
2	1.00	1.00	1.00	50
micro avg	1.00	1.00	1.00	150
macro avg	1.00	1.00	1.00	150
weighted avg	1.00	1.00	1.00	150

```

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]

```

4.1 References

1. <https://machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/>