COMPARITIVE ANALYSIS FOR CLASSIFICATION ALGORITHMS

##### A MINI PROJECT REPORT

###### ***Submitted by***

##### SURABHI AGRAWAL (RA1511003010674)

**S.ADITYA(RA1511003010222)**

***for the course***

**15IT322E – Python Programming**

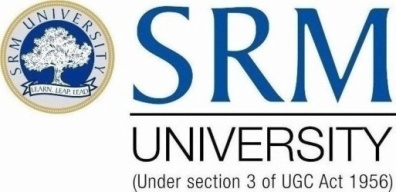
***in partial fulfillment for the award of the degree***

***of***

**Bachelor of Technology**

IN

# COMPUTER SCIENCE AND ENGINEERING



SRM UNIVERSITY

KATTANKULATHUR

##### NOVEMBER 2017

SRM UNIVERSITY

KATTANKULATHUR

**BONAFIDE CERTIFICATE**

Certified that this project report “COMPARITIVE ANALYSIS FOR CLASSIFICATION ALGORITHMS ” is the bonafide work of “SURABHI AGRAWAL (RA1511003010674), S. ADITYA(RA1511003010222)” who carried out the project work as part of their course 15IT322E – Python Programming

**SIGNATURE SIGNATURE**

**Course Instructor** **Dr. B. Amutha**

**HEAD OF THE DEPARTMENT Computer Science & Engg**

**INTERNAL EXAMINER**

Abstract

Our project aims at using various classification techniques to classify the iris data set as well as the diabetes data set according to the features to obtain maximum accuracy. We will use KNN classification,Logistic Regression, Support Vector Machine and CART to classify the same data set to check the efficiency of the algorithm as well as to compare the accuracy of the results obtained.

In [machine learning](https://en.wikipedia.org/wiki/Machine_learning), support vector machines (SVMs, also support vector networks) are [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) models with associated learning [algorithms](https://en.wikipedia.org/wiki/Algorithm) that analyze data used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression analysis](https://en.wikipedia.org/wiki/Regression_analysis). Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other.

*k*-NN is a type of [instance-based learning](https://en.wikipedia.org/wiki/Instance-based_learning), or [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning), where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms.

The binary logistic model, Logistic regression is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the odds of a given outcome by a specific factor.

Classification and regression trees (CART) are a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics) [decision tree learning](https://en.wikipedia.org/wiki/Decision_tree_learning) technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively.

[Decision trees](https://en.wikipedia.org/wiki/Decision_trees) are formed by a collection of rules based on variables in the modeling data set

Introduction

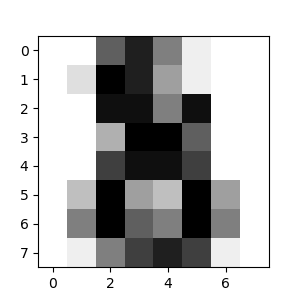
Iris Data Set:

The Iris flower data set or Fisher's Iris data set is a [multivariate](https://en.wikipedia.org/wiki/Multivariate_statistics) [data set](https://en.wikipedia.org/wiki/Data_set) introduced by the British [statistician](https://en.wikipedia.org/wiki/Statistician) and [biologist](https://en.wikipedia.org/wiki/Biologist) [Ronald Fisher](https://en.wikipedia.org/wiki/Ronald_Fisher) in his 1936 paper The use of multiple measurements in taxonomic problems as an example of [linear discriminant analysis](https://en.wikipedia.org/wiki/Linear_discriminant_analysis).[[1]](https://en.wikipedia.org/wiki/Iris_flower_data_set#cite_note-fisher36-1) It is sometimes called Anderson's Iris data set because [Edgar Anderson](https://en.wikipedia.org/wiki/Edgar_Anderson) collected the data to quantify the [morphologic](https://en.wikipedia.org/wiki/Morphology_(biology)) variation of [Iris](https://en.wikipedia.org/wiki/Iris_(plant)) flowers of three related species.

The data set consists of 50 samples from each of three species of Iris ([Iris setosa](https://en.wikipedia.org/wiki/Iris_setosa), [Iris virginica](https://en.wikipedia.org/wiki/Iris_virginica) and [Iris versicolor](https://en.wikipedia.org/wiki/Iris_versicolor)). Four [features](https://en.wikipedia.org/wiki/Features_(pattern_recognition)) were measured from each sample: the length and the width of the [sepals](https://en.wikipedia.org/wiki/Sepal) and [petals](https://en.wikipedia.org/wiki/Petal), in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Digits Data Set:

This dataset is made up of 1797 8x8 images. Each image, like the one shown below, is of a hand-written digit. In order to utilize an 8x8 figure like this, we’d have to first transform it into a feature vector with length 64.



We create a digit database by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training, cross-validation and writer dependent testing, and the digits written by the other 14 are used for writer independent testing. This database is also available in the UNIPEN format.

These writers are asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution. Subject are monitored only during the first entry screens. Each screen contains five boxes with the digits to be written displayed above. Subjects are told to write only inside these boxes. If they make a mistake or are unhappy with their writing, they are instructed to clear the content of a box by using an on-screen button. The first ten digits are ignored because most writers are not familiar with this type of input devices, but subjects are not aware of this.

SVM:

[Classifying data](https://en.wikipedia.org/wiki/Statistical_classification) is a common task in [machine learning](https://en.wikipedia.org/wiki/Machine_learning). Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a p {\displaystyle p}pp-dimensional vector (a list of p {\displaystyle p}pnumbers), and we want to know whether we can separate such points with a (p-1) {\displaystyle (p-1)}-dimensional [hyperplane](https://en.wikipedia.org/wiki/Hyperplane). This is called a [linear classifier](https://en.wikipedia.org/wiki/Linear_classifier).

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the [kernel trick](https://en.wikipedia.org/wiki/Kernel_trick), implicitly mapping their inputs into high-dimensional feature spaces.

When data are not labeled, supervised learning is not possible, and an [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning) approach is required, which attempts to find natural [clustering of the data](https://en.wikipedia.org/wiki/Data_clustering) to groups, and then map new data to these formed groups. The clustering algorithm which provides an improvement to the support vector machines is called support vector clustering and is oftenused in industrial applications either when data are not labeled or when only some data are labeled as a preprocessing for a classification pass.

KNN:

The naive version of the algorithm is easy to implement by computing the distances from the test example to all stored examples, but it is computationally intensive for large training sets. Using an approximate [nearest neighbor search](https://en.wikipedia.org/wiki/Nearest_neighbor_search) algorithm makes k-NN computationally tractable even for large data sets.

Logistic Regression:

Logistic regression can be binomial, ordinal or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a [dependent variable](https://en.wikipedia.org/wiki/Dependent_variable) can have only two possible types, "0" and "1" (which may represent, for example, "dead" vs. "alive" or "win" vs. "loss"). [Multinomial logistic regression](https://en.wikipedia.org/wiki/Multinomial_logit) deals with situations where the outcome can have three or more possible types (e.g., "disease A" vs. "disease B" vs. "disease C") that are not ordered. [Ordinal logistic regression](https://en.wikipedia.org/wiki/Ordinal_logistic_regression) deals with dependent variables that are ordered.

CART:

[Decision trees](https://en.wikipedia.org/wiki/Decision_trees) are formed by a collection of rules based on variables in the modeling data set:

* Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable
* Once a rule is selected and splits a node into two, the same process is applied to each "child" node (i.e. it is a recursive procedure)
* Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met. (Alternatively, the data are split as much as possible and then the tree is later [pruned](https://en.wikipedia.org/wiki/Pruning_(decision_trees)).)

Each branch of the tree ends in a terminal node. Each observation falls into one and exactly one terminal node, and each terminal node is uniquely defined by a set of rules.

Related Work

IRIS DATA SET:

1. R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". [*Annals of Eugenics*](https://en.wikipedia.org/wiki/Annals_of_Eugenics). 7 (2): 179–188. [*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier):[*10.1111/j.1469-1809.1936.tb02137.x*](https://doi.org/10.1111%2Fj.1469-1809.1936.tb02137.x).
2. Edgar Anderson (1936). [*"The species problem in Iris"*](http://biostor.org/reference/11559). [*Annals of the Missouri Botanical Garden*](https://en.wikipedia.org/wiki/Annals_of_the_Missouri_Botanical_Garden). 23 (3): 457–509. [*JSTOR*](https://en.wikipedia.org/wiki/JSTOR) [*2394164*](https://www.jstor.org/stable/2394164).

**LOGISTIC REGRESSION:**

1. [David A. Freedman](https://en.wikipedia.org/wiki/David_A._Freedman) (2009). Statistical Models: Theory and Practice. [Cambridge University Press](https://en.wikipedia.org/wiki/Cambridge_University_Press). p. 128.
2. ^ [Jump up to:a](https://en.wikipedia.org/wiki/Logistic_regression#cite_ref-wal67est_2-0) [b](https://en.wikipedia.org/wiki/Logistic_regression#cite_ref-wal67est_2-1) Walker, SH; Duncan, DB (1967). "Estimation of the probability of an event as a function of several independent variables". Biometrika. 54: 167–178. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.2307/2333860](https://doi.org/10.2307%2F2333860).
3. Cox, DR (1958). "The regression analysis of binary sequences (with discussion)". J Roy Stat Soc B. 20: 215–242. [JSTOR](https://en.wikipedia.org/wiki/JSTOR) [2983890](https://www.jstor.org/stable/2983890).
4. Boyd, C. R.; Tolson, M. A.; Copes, W. S. (1987). "Evaluating trauma care: The TRISS method. Trauma Score and the Injury Severity Score". The Journal of trauma. 27 (4): 370–378. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1097/00005373-198704000-00005](https://doi.org/10.1097%2F00005373-198704000-00005). [PMID](https://en.wikipedia.org/wiki/PubMed_Identifier) [3106646](https://www.ncbi.nlm.nih.gov/pubmed/3106646).
5. Kologlu M., Elker D., Altun H., Sayek I. Validation of MPI and OIA II in two different groups of patients with secondary peritonitis // Hepato-Gastroenterology. – 2001. – Vol. 48, № 37. – pp. 147–151.
6. Biondo S., Ramos E., Deiros M. et al. Prognostic factors for mortality in left colonic peritonitis: a new scoring system // J. Am. Coll. Surg. – 2000. – Vol. 191, № 6. – Р. 635-642.

CART:

1. [*"Horse Drawn Carriages"*](http://www.scalemodelhorsedrawnvehicle.co.uk/(carriages).htm). Scalemodelhorsedrawnvehicle.co.uk*. Retrieved 2014-08-25*.
2. [*"Horse drawn vehicles in the 19th Century - Driffield Post Times"*](http://www.driffieldtoday.co.uk/news/local/horse-drawn-vehicles-in-the-19th-century-1-4189245). Driffieldtoday.co.uk. 2012-01-27*. Retrieved 2014-08-25*.

Individual Contribution

Surabhi Agrawal:

For Iris Data Set, applied classification functions from sklearn module and compared the different model’s efficiency along with project documentation.

S.Aditya:

For Handwritten Digits Recognition Data Set, applied various built in classification methods from sklearn module and compared their efficiency. Also, built a KNN classifier from scratch and compare the efficiency of the classifier with the built in KNN classifier for Iris Data Set as well as Handwritten Digit Recognition Data Set.

Algorithm

KNN Algorithm:

### k-nearest neighbor (Knn) algorithm pseudocode:

Let (Xi, Ci) where i = 1, 2……., n be data points. Xi denotes feature values & Ci denotes labels for Xi for each i.  
Assuming the number of classes as ‘c’  
Ci ∈ {1, 2, 3, ……, c} for all values of i

Let x be a point for which label is not known, and we would like to find the label class using k-nearest neighbor algorithms.

### Knn Algorithm Pseudocode:

1. Calculate “d(x, xi)” i =1, 2, ….., **n**; where **d** denotes the [Euclidean distance](https://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/) between the points.
2. Arrange the calculated **n** Euclidean distances in non-decreasing order.
3. Let **k** be a +ve integer, take the first **k** distances from this sorted list.
4. Find those **k**-points corresponding to these **k**-distances.
5. Let **k**i denotes the number of points belonging to the ith class among **k** points i.e. k ≥ 0
6. If ki >kj ∀ i ≠ j then put x in class i.

SVM Algorithm:

Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However,  it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_1.png)

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

The process of segregating the two classes with a hyper-plane. “How can we identify the right hyper-plane.

* **Identify the right hyper-plane:** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.  
  You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.



* **Identify the right hyper-plane:** Here, we have three hyper-planes (A, B and C) and all are segregating the classes well.
* 

Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. [[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_4.png)](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_4.png)Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

* **Identify the right hyper-plane**

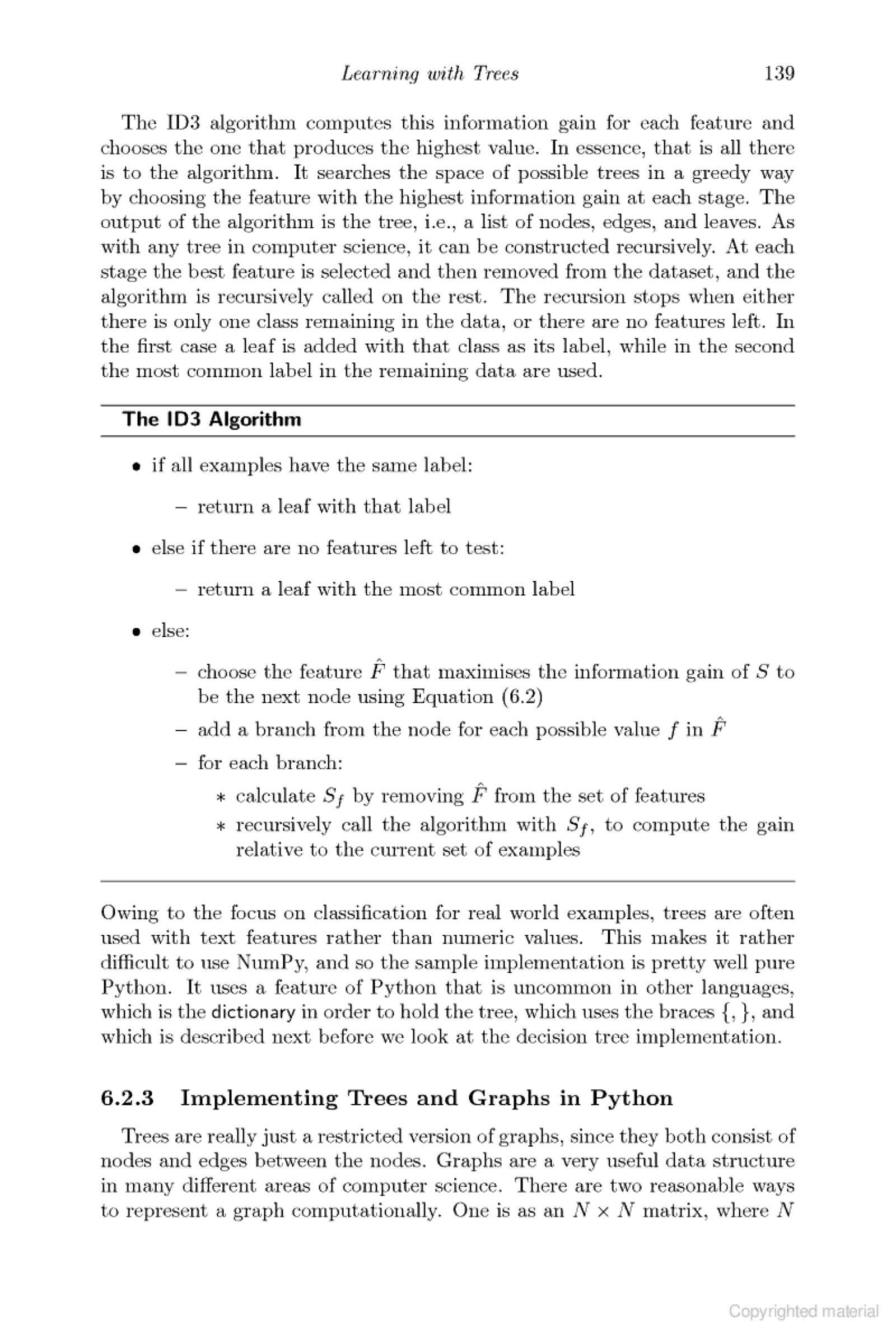
Some of you may have selected the hyper-plane **B**as it has higher margin compared to **A.**But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

**[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_5.png)**

* **Can we classify two classes :** Below, segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.  **[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_61.png)**
* One star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.  
  **[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_71.png)**
* **Find the hyper-plane to segregate to classes:**In the scenario below, we can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes.**[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_8.png)**
* SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Plotting the data points on axis x and z:  
  [[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_9.png)](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_9.png)In above plot, points to consider are:
  + All values for z would be positive always because z is the squared sum of both x and y
  + In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the [**kernel**](https://en.wikipedia.org/wiki/Kernel_method)**trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you’ve defined.

CART:



Program

IRIS DATA SET:

# coding: utf-8

# In[2]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

features = iris.feature\_names

f = plt.figure(figsize=(10,12))

gs = gridspec.GridSpec(3,2)

r,c = 0,0

for i in range(4):

for j in range(i+1,4):

if c == 2:

r += 1

c = 0

x\_min, x\_max = X[:,i].min() - 0.5, X[:,i].max() + 0.5

y\_min, y\_max = X[:,j].min() - 0.5, X[:,j].max() + 0.5

ax = plt.subplot(gs[r,c])

ax.scatter(X[:,i], X[:,j], c=y, edgecolors='k', cmap=plt.cm.Paired)

ax.set\_xlim(x\_min,x\_max)

ax.set\_ylim(y\_min,y\_max)

ax.set\_xlabel(features[i])

ax.set\_ylabel(features[j])

ax.set\_xticklabels([])

ax.set\_yticklabels([])

f.add\_subplot(ax)

c += 1

f.subplots\_adjust(hspace=2, wspace=1)

gs.tight\_layout(f, rect=[0,0,1,0.95])

plt.suptitle('Iris Data Visualisation', fontsize=20)

plt.show()

path = r'C:\Users\Lenovo\AppData\Local\Programs\Python\Python35\iris\_vis.png'

import os

if not os.path.isfile(path):

f.savefig('iris\_vis.png')

# In[3]:

np.random.seed(0)

indices = np.random.permutation(len(X))

X\_train = X[indices[:-10]]

y\_train = y[indices[:-10]]

X\_test = X[indices[-10:]]

y\_test = y[indices[-10:]]

# In[4]:

def make\_grid(X,Y,feature,model):

h = .02

model.fit(X, Y)

x\_min, x\_max = X[:, 0].min() - .5, X[:, 0].max() + .5

y\_min, y\_max = X[:, 1].min() - .5, X[:, 1].max() + .5

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

return (Z, xx, yy)

# In[5]:

def classify(model,title,filename):

features = iris['feature\_names']

f = plt.figure(figsize=(10,12))

gs = gridspec.GridSpec(3,2)

r,c = 0,0

acc = [[],[]]

for i in range(4):

for j in range(i+1,4):

x = np.stack((X\_train[:,i],X\_train[:,j]),axis = 1)

(Z, xx, yy) = make\_grid(x,y\_train,[features[i],features[j]],model)

model.fit(x, y\_train)

z = model.predict(x)

acc[0].append(accuracy\_score(z, y\_train))

xt = np.stack((X\_test[:,i],X\_test[:,j]),axis = 1)

z = model.predict(xt)

acc[1].append(accuracy\_score(z, y\_test))

if c == 2:

r += 1

c = 0

ax = plt.subplot(gs[r,c])

ax.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

ax.scatter(x[:,0], x[:,1], c=y\_train, edgecolors='k', cmap=plt.cm.Paired)

ax.set\_xlim(xx.min(), xx.max())

ax.set\_ylim(yy.min(), yy.max())

ax.set\_xlabel(features[i])

ax.set\_ylabel(features[j])

ax.set\_xticklabels([])

ax.set\_yticklabels([])

f.add\_subplot(ax)

c += 1

f.subplots\_adjust(hspace=2, wspace=1)

gs.tight\_layout(f, rect=[0,0,1,0.95])

plt.suptitle(title, fontsize=20)

plt.show()

path = r'C:\Users\Lenovo\AppData\Local\Programs\Python\Python35\%s'%(filename)

import os

if not os.path.isfile(path):

f.savefig(filename)

error = [sum(acc[i])/len(acc[i]) \*100 for i in range(2)]

print(model)

print('Error rate for training data : %d%%'%(error[0]))

print('Error rate for testing data : %d%%\n'%(error[1]))

stats.update({title:error})

# In[8]:

from sklearn.metrics import accuracy\_score

stats = {}

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=1e5)

title = 'Logistic Regression'

filename = 'iris\_logreg.png'

classify(model,title,filename)

# In[9]:

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=3)

title = 'KNN k=3'

filename = 'iris\_knn3.png'

classify(model,title,filename)

# In[10]:

model = KNeighborsClassifier(n\_neighbors=7)

title = 'KNN k=7'

filename = 'iris\_knn7.png'

classify(model,title,filename)

# In[11]:

from sklearn.svm import SVC

model = SVC(kernel='linear', C=1.0)

title = 'SVC linear'

filename = 'iris\_svm\_linear.png'

classify(model,title,filename)

# In[12]:

model = SVC(kernel='rbf', gamma=0.7, C=1.0)

title = 'SVM RBF'

filename = 'iris\_svm\_rbf.png'

classify(model,title,filename)

# In[13]:

model = SVC(kernel='poly', degree=3, C=1.0)

title = 'SVM poly(degree=3)'

filename = 'iris\_svm\_poly.png'

classify(model,title,filename)

# In[14]:

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

title = 'CART'

filename = 'iris\_cart.png'

classify(model,title,filename)

# In[15]:

k, v = list(stats.keys()), np.asarray(list(stats.values()))

stat = pd.DataFrame({'Classifiers':k, 'Training Accuracy':v[:,0], 'Testing Accuracy':v[:,1]})

print(stat)

# In[16]:

n\_groups = 7

fig, ax = plt.subplots()

index = np.arange(n\_groups)

bar\_width = 0.35

opacity = 0.8

rects1 = plt.bar(index, v[:,0], bar\_width,

alpha=opacity,

color='b',

label='Training')

rects2 = plt.bar(index + bar\_width, v[:,1], bar\_width,

alpha=opacity,

color='g',

label='GTesting')

plt.xlabel('Models')

plt.ylabel('Accuracy (%)')

plt.title('Accuracy Comparison')

plt.xticks(index + bar\_width, k)

plt.legend()

plt.tight\_layout()

plt.show()

IRIS KNN:

# coding: utf-8

# In[1]:

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score

from collections import Counter

def predict(X\_train, y\_train, x\_test, k):

distances = []

targets = []

for i in range(len(X\_train)):

distance = np.sqrt(np.sum(np.square(x\_test - X\_train[i, :])))

distances.append([distance, i])

distances = sorted(distances)

for i in range(k):

index = distances[i][1]

targets.append(y\_train[index])

return Counter(targets).most\_common(1)[0][0]

# In[2]:

def knn(X\_train, y\_train, X\_test, z, k):

if k > len(X\_train):

raise ValueError

for i in range(len(X\_test)):

z.append(predict(X\_train, y\_train, X\_test[i, :], k))

# In[3]:

iris = load\_iris()

X = iris.data

y = iris.target

np.random.seed(0)

indices = np.random.permutation(len(X))

X\_train = X[indices[:-20]]

y\_train = y[indices[:-20]]

X\_test = X[indices[-20:]]

y\_test = y[indices[-20:]]

# In[4]:

z = []

try:

knn(X\_train, y\_train, X\_test, z, 7)

z = np.asarray(z)

accuracy = accuracy\_score(y\_test, z) \* 100

print('Accuracy of self made KNN: %d%%' %accuracy)

except ValueError:

print('Cannot have more neighbors than training samples.')

# In[5]:

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=7)

model.fit(X\_train,y\_train)

pred = model.predict(X\_test)

acc = accuracy\_score(y\_test,pred) \*100

print('Accuracy of sklearn KNN : %d%%'%acc)

DIGITS DATA SET:

# coding: utf-8

# In[1]:

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

from sklearn.datasets import load\_digits

from sklearn.metrics import accuracy\_score

dig = load\_digits()

X = dig.data

y = dig.target

np.random.seed(0)

t = len(X)//10

indices = np.random.permutation(len(X))

X\_train = X[indices[:-t]]

y\_train = y[indices[:-t]]

X\_test = X[indices[-t:]]

y\_test = y[indices[-t:]]

acc = []

model\_names = ['Logistic Regression', 'KNN k=3', 'KNN k=7', 'SVM Linear', 'SVM RBF', 'SVM polynomial', 'CART']

# In[2]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=1e5)

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for logistic regression: %d%%\n'%(acc[0]))

# In[3]:

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=3)

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for KNN classifier with k=3: %d%%\n'%(acc[1]))

# In[4]:

model = KNeighborsClassifier(n\_neighbors=7)

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for KNN classifier with k=7: %d%%\n'%(acc[2]))

# In[5]:

from sklearn.svm import SVC

model = SVC(kernel='linear',C=1.0)

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for SVM-linear classifier: %d%%\n'%(acc[3]))

# In[6]:

model = SVC(kernel='rbf',gamma=0.001,C=1.0)

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for SVM-rbf classifier: %d%%\n'%(acc[4]))

# In[7]:

model = SVC(kernel='poly', degree=3, C=1.0)

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for SVM-linear classifier: %d%%\n'%(acc[5]))

# In[8]:

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

model.fit(X\_train,y\_train)

z = model.predict(X\_test)

acc.append(accuracy\_score(z, y\_test)\*100)

print(model)

print('Accuracy for CART: %d%%'%(acc[6]))

# In[14]:

import pandas as pd

accuracy = pd.DataFrame({'Classifier':model\_names, 'Accuracy':acc})

print(accuracy)

# In[18]:

n\_groups = 7

fig, ax = plt.subplots()

index = np.arange(n\_groups)

bar\_width = 0.35

opacity = 0.8

rects = plt.bar(index, acc, bar\_width,

alpha=opacity,

color='b', label='Accuracy')

plt.xlabel('Models')

plt.ylabel('Accuracy (%)')

plt.title('Accuracy Comparison')

plt.xticks(index + bar\_width, model\_names)

plt.legend()

plt.tight\_layout()

plt.show()

DIGITS\_KNN.PY

# coding: utf-8

# In[1]:

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

from sklearn.datasets import load\_digits

from sklearn.metrics import accuracy\_score

from collections import Counter

def predict(X\_train, y\_train, x\_test, k):

distances = []

targets = []

for i in range(len(X\_train)):

distance = np.sqrt(np.sum(np.square(x\_test - X\_train[i, :])))

distances.append([distance, i])

distances = sorted(distances)

for i in range(k):

index = distances[i][1]

targets.append(y\_train[index])

return Counter(targets).most\_common(1)[0][0]

# In[2]:

def knn(X\_train, y\_train, X\_test, z, k):

if k > len(X\_train):

raise ValueError

for i in range(len(X\_test)):

z.append(predict(X\_train, y\_train, X\_test[i, :], k))

# In[3]:

dig = load\_digits()

X = dig.data

y = dig.target

np.random.seed(0)

indices = np.random.permutation(len(X))

X\_train = X[indices[:-200]]

y\_train = y[indices[:-200]]

X\_test = X[indices[-200:]]

y\_test = y[indices[-200:]]

# In[4]:

z = []

try:

knn(X\_train, y\_train, X\_test, z, 7)

z = np.asarray(z)

accuracy = accuracy\_score(y\_test, z) \* 100

print('Accuracy of self made KNN: %d%%' %accuracy)

except ValueError:

print('Cannot have more neighbors than training samples.')

# In[5]:

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=7)

model.fit(X\_train,y\_train)

pred = model.predict(X\_test)

acc = accuracy\_score(y\_test,pred) \*100

print('Accuracy of sklearn KNN : %d%%'%acc)

# In[6]:

from sklearn import svm

def makearr(i):

arr = np.array(X[i],ndmin=2)

return arr

model = svm.SVC(gamma=0.0001,C=10)

model.fit(X\_train,y\_train)

i = int(input('Enter the data to be predicted : '))

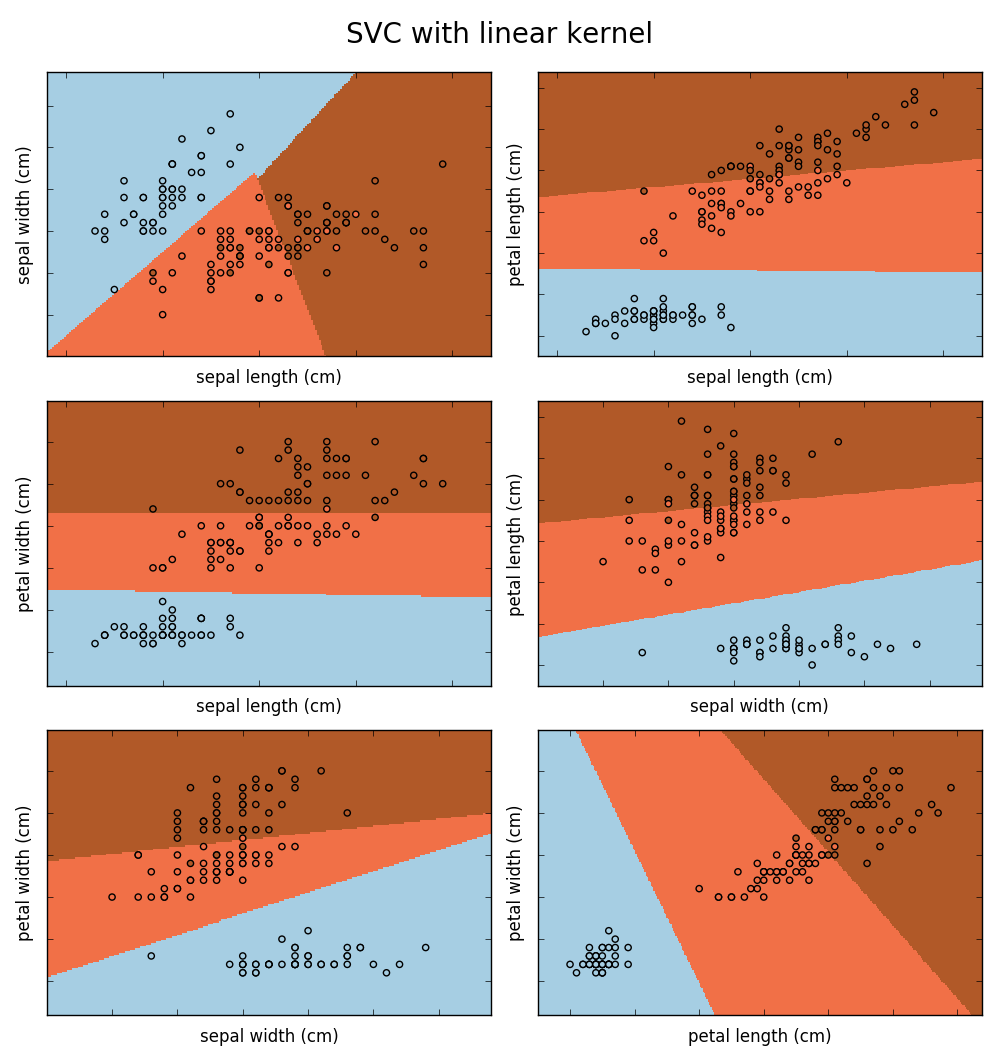
a = makearr(i)

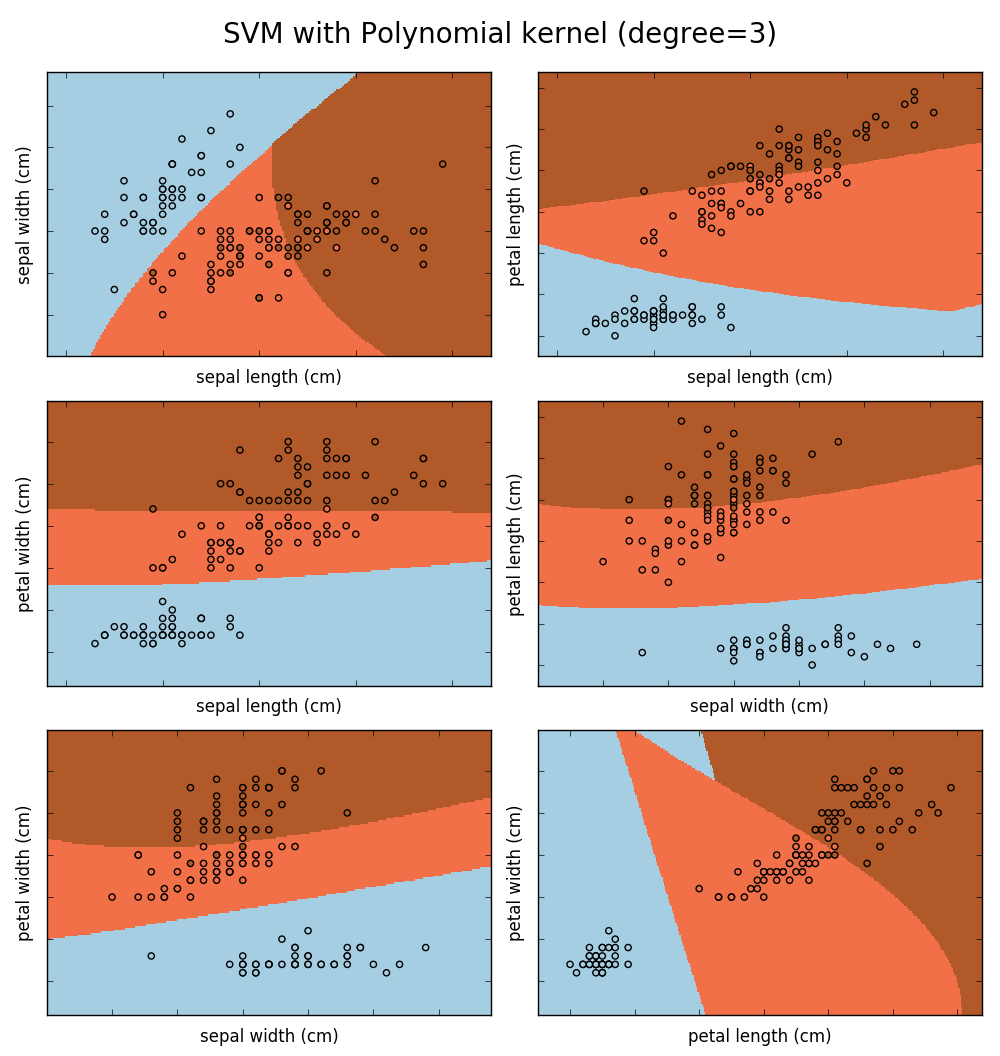
print(model.predict(a))

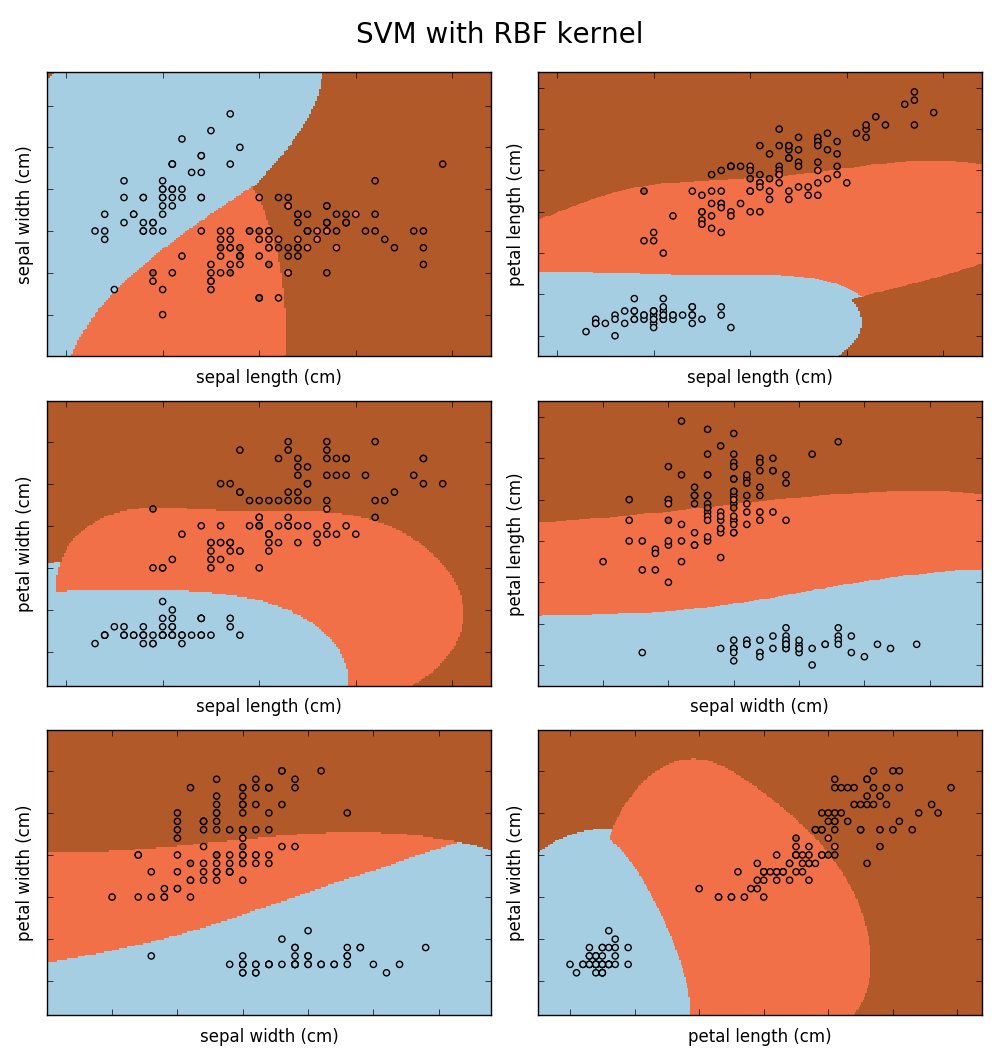
plt.imshow(dig.images[i], cmap=plt.cm.gray\_r, interpolation='nearest')

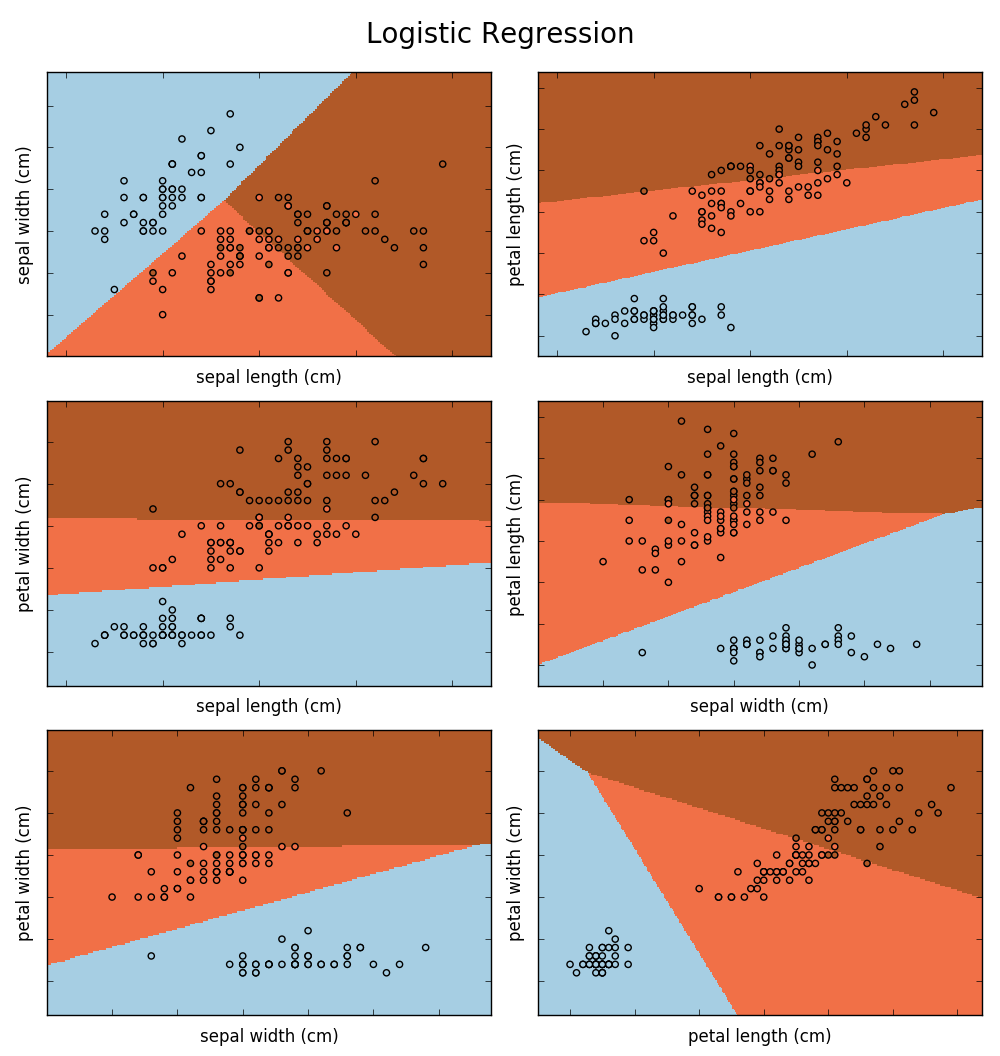
plt.show()

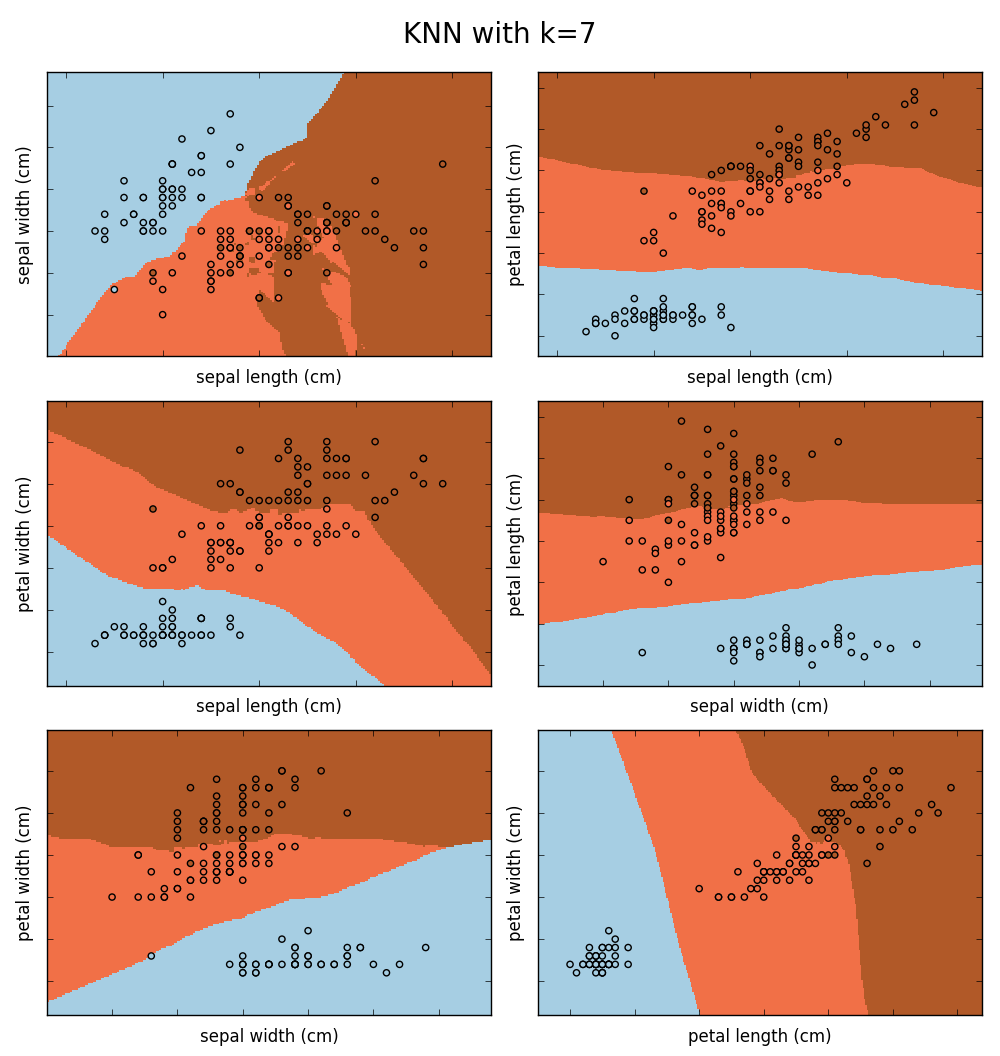
Results

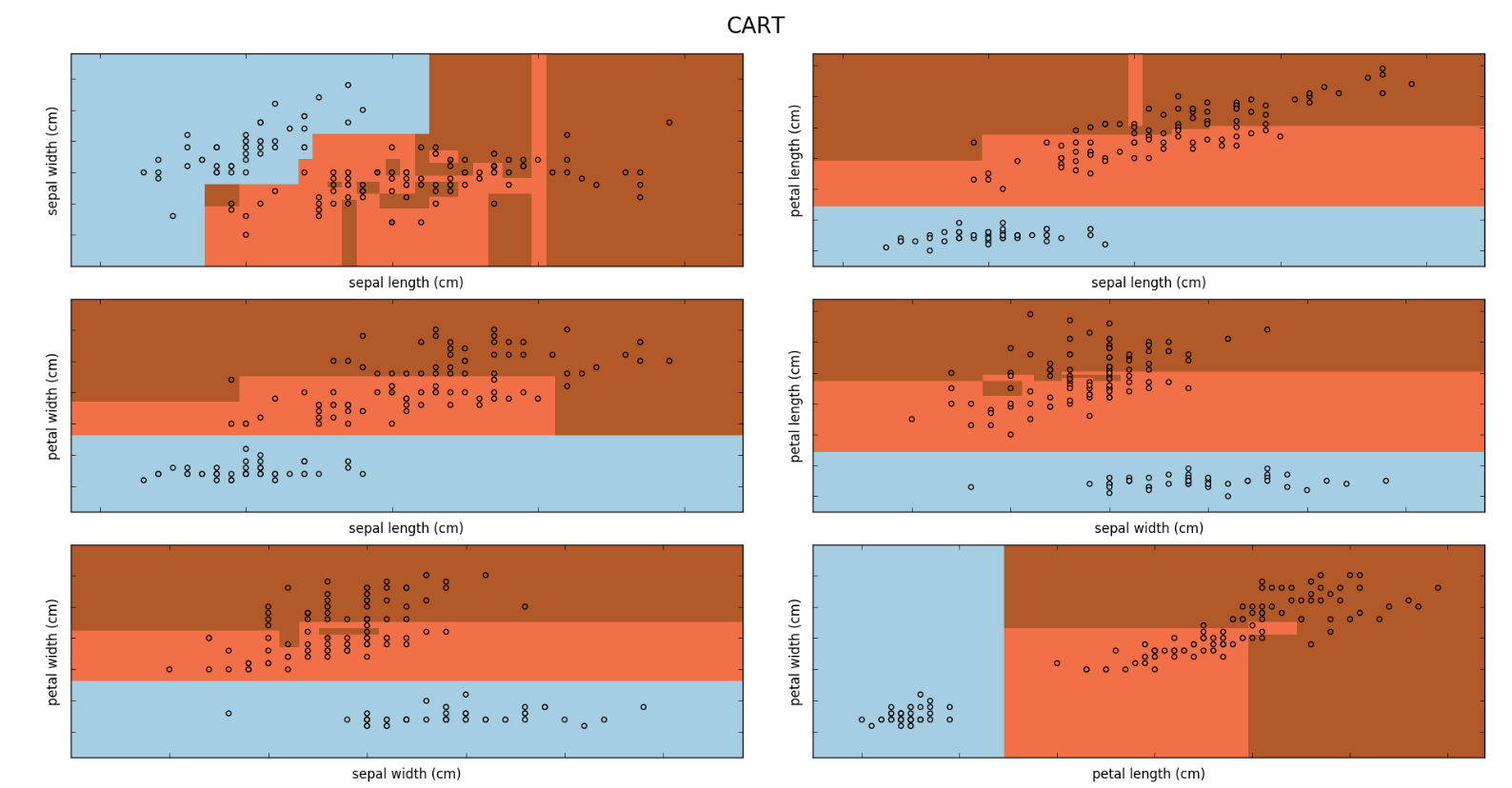


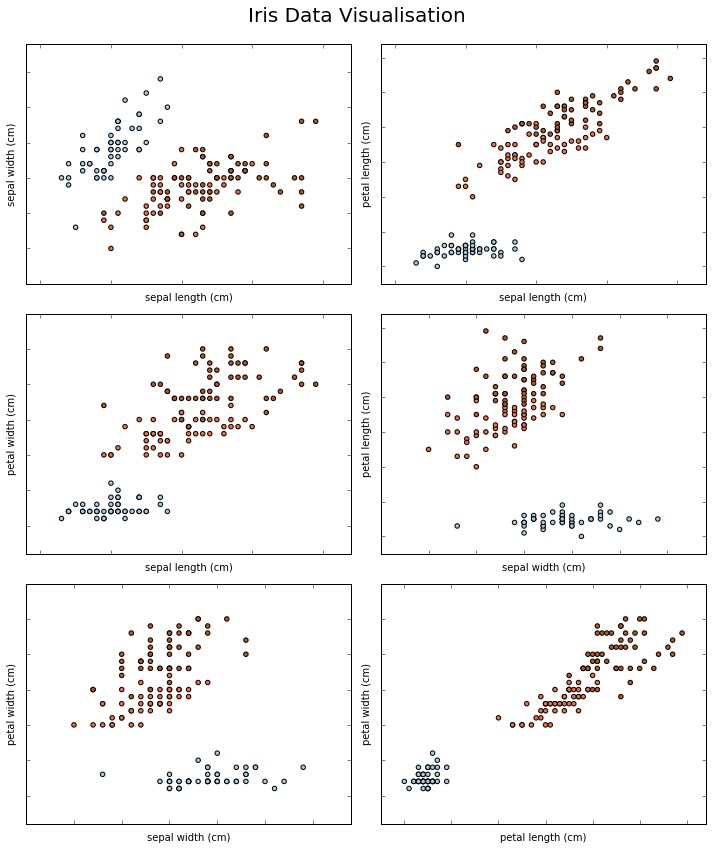












OUTPUT FOR IRIS.PY

LogisticRegression(C=100000.0, class\_weight=None, dual=False,

fit\_intercept=True, intercept\_scaling=1, max\_iter=100,

multi\_class='ovr', n\_jobs=1, penalty='l2', random\_state=None,

solver='liblinear', tol=0.0001, verbose=0, warm\_start=False)

Accuracy for logistic regression: 93%

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=3, p=2,

weights='uniform')

Accuracy for KNN classifier with k=3: 99%

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=7, p=2,

weights='uniform')

Accuracy for KNN classifier with k=7: 100%

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='linear',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

Accuracy for SVM-linear classifier: 98%

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma=0.001, kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

Accuracy for SVM-rbf classifier: 100%

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='poly',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

Accuracy for SVM-linear classifier: 100%

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

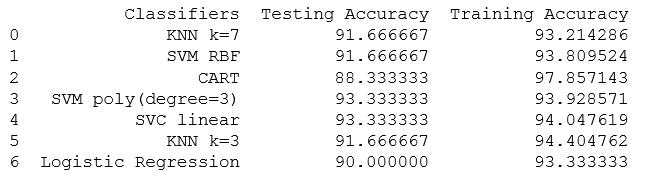
min\_impurity\_decrease=0.0, min\_impurity\_split=None,

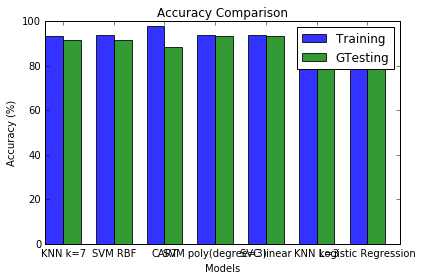
min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,

splitter='best')

Accuracy for CART: 85%



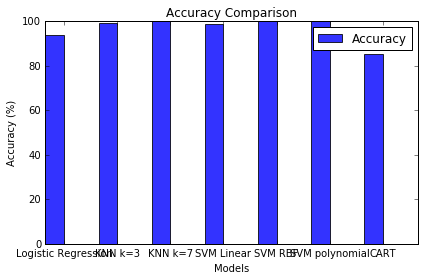
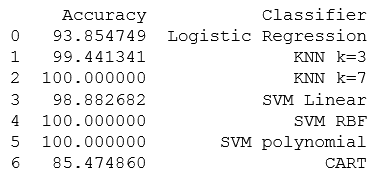


OUTPUT FOR IRIS\_KNN.PY

Accuracy of self made KNN: 85%

Accuracy of sklearn KNN: 85%

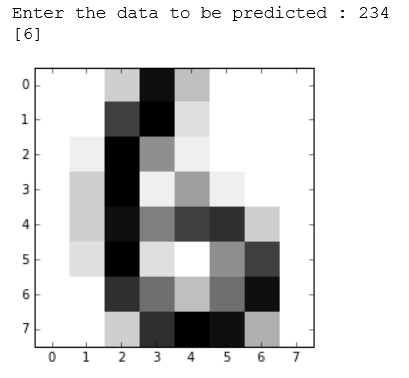
OUTPUT FOR DIGITSALL.PY



OUTPUT FOR DIGITS.PY

Accuracy for self made KNN: 99%

Accuracy for sklearn KNN: 99%



Conclusion

The classification techniques used namely, Support Vector Machines, KNN, Logistic Regression and CART which were used to classify the two data sets namely Iris Data Set and Digits Data Set yielded various outputs with similar accuracies for the testing and training data.

The following observations were made:

1. For the Data Sets, when the following built in classifiers were applied:

* Logistic Regression
* K Nearest Neighbors Classifier with k=3
* K Nearest Neighbors Classifier with k=7
* Support Vector Machine with Linear Kernel
* Support Vector Machine with Radial Base Function Kernel(RBF)
* Support Vector Machine with Polynomial Kernel(Degree=3)
* Classification and Regression Tree (CART)

And, we conclude that for the Handwritten Digit Recognition Data Set, KNN for k=7, SVM with RBF as well as Polynomial Kernel performed with maximum efficiency of 100% for Testing data being 10% the size of training data.

For Iris Data Set, the CART classifier performed with maximum efficiency of 97.8% for the testing data being 15% of the training data.

1. For the self-built KNN classifier, we observed that it performed with the same efficiency as the built in classifier from sklearn module. For Iris Data Set both the the classifier performed with an efficiency of 85% and for the Handwritten Digit Recognition Data Set, both performed with the efficiency of 99%.

References

1. [www.tutorialspoint.com](http://www.tutorialspoint.com)
2. [www.wikepedia.org](http://www.wikepedia.org)
3. <https://machinelearningmastery.com/spot-check-classification-machine-learning-algorithms-python-scikit-learn/>
4. <http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits>
5. <http://dataaspirant.com/2016/12/23/k-nearest-neighbor-classifier-intro/>
6. <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>
7. <http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression/>

**GitHub Link for project:** <https://github.com/surabhi711/pythonProject>