**Classification**

The goal of text classifications to automatically classify the text/numbers documents(data) into one or more defined categories.

Text Classification is an example of supervised machine learning task since a labelled dataset containing text documents and their labels is used for train a classifier.

An end-to-end text classification pipeline is composed of three main components:

**1. Dataset Preparation:**The first step is the Dataset Preparation step which includes the process of loading a dataset and performing basic pre-processing. The dataset is then split into train and validation sets.  
**2. Feature Engineering:**The next step is the Feature Engineering in which the raw dataset is transformed into flat features which can be used in a machine learning model. This step also includes the process of creating new features from the existing data.  
**3. Model Training:**The final step is the Model Building step in which a machine learning model is trained on a labelled dataset.

**4. Improve Performance of Text Classifier:**In this article, we will also look at the different ways to improve the performance of text classifiers.

**Data**

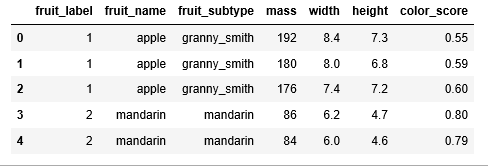
Dataset contains few dozen oranges, lemons and apples of different varieties, and recorded their measurements in a table.

%matplotlib inline

import pandas as pd

import matplotlib.pyplot as plt

fruits = pd.read\_table('fruit\_data\_with\_colors.txt')  
fruits.head()



Each row of the dataset represents one piece of the fruit as represented by several features that are in the table’s columns.

We have 59 pieces of fruits and 7 features in the dataset:

print(fruits.shape)

***(59, 7)***

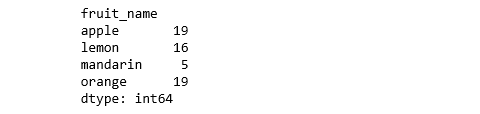
We have four types of fruits in the dataset:

print(fruits['fruit\_name'].unique())

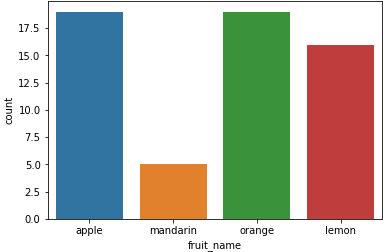
***[‘apple’ ‘mandarin’ ‘orange’ ‘lemon’]***

The data is pretty balanced except mandarin. We will just have to go with it.

print(fruits.groupby('fruit\_name').size())



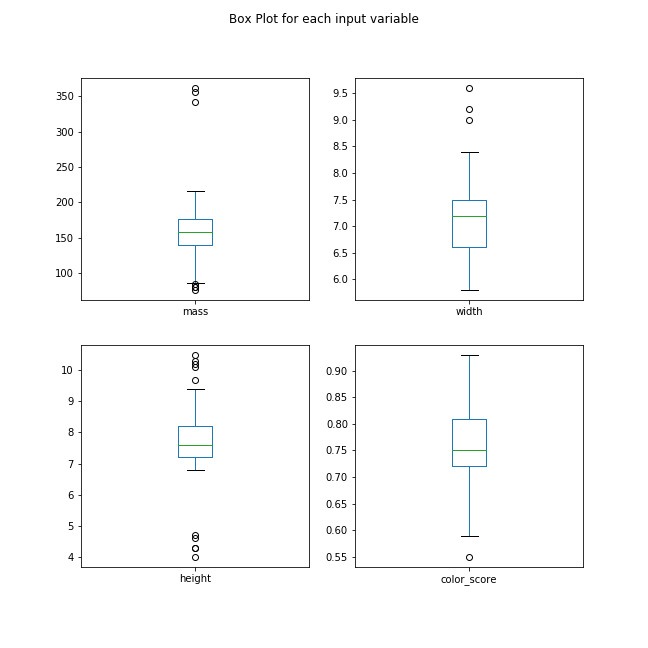
import seaborn as sns  
sns.countplot(fruits['fruit\_name'],label="Count")  
plt.show()



**Visualization**

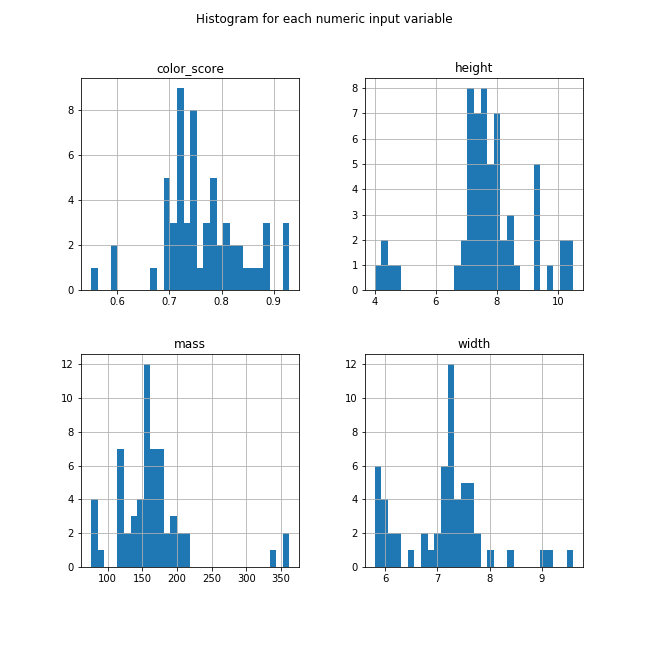
* Box plot for each numeric variable will give us a clearer idea of the distribution of the input variables:

fruits.drop('fruit\_label', axis=1).plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False, figsize=(9,9),   
 title='Box Plot for each input variable')  
plt.savefig('fruits\_box')  
plt.show()



* It looks like perhaps color score has a near Gaussian distribution.

import pylab as pl  
fruits.drop('fruit\_label' ,axis=1).hist(bins=30, figsize=(9,9))  
pl.suptitle("Histogram for each numeric input variable")  
plt.savefig('fruits\_hist')  
plt.show()

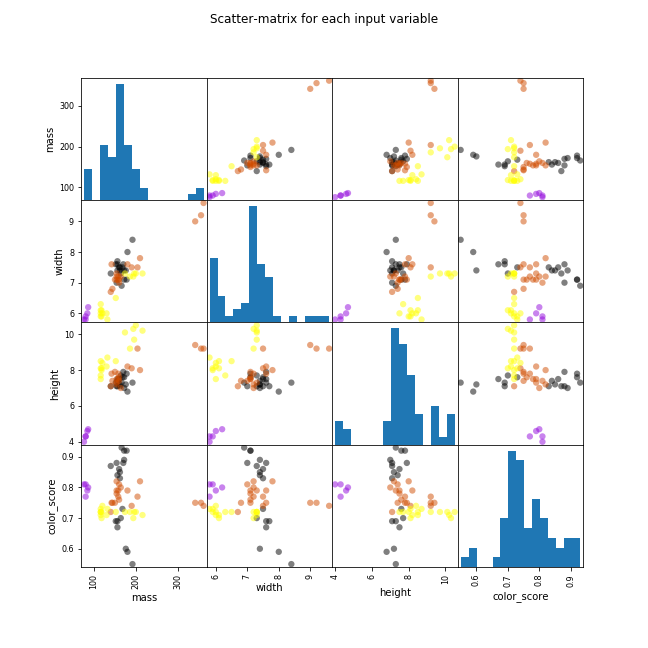


* Some pairs of attributes are correlated (mass and width). This suggests a high correlation and a predictable relationship.

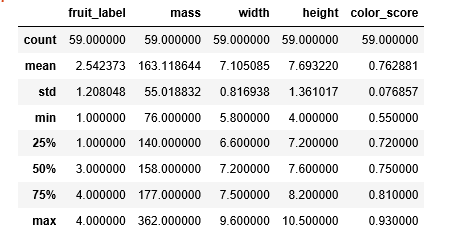
from pandas.tools.plotting import scatter\_matrix  
from matplotlib import cm

feature\_names = ['mass', 'width', 'height', 'color\_score']  
X = fruits[feature\_names]  
y = fruits['fruit\_label']

cmap = cm.get\_cmap('gnuplot')  
scatter = pd.scatter\_matrix(X, c = y, marker = 'o', s=40, hist\_kwds={'bins':15}, figsize=(9,9), cmap = cmap)  
plt.suptitle('Scatter-matrix for each input variable')  
plt.savefig('fruits\_scatter\_matrix')



**Statistical Summary**



We can see that the numerical values do not have the same scale. We will need to apply scaling to the test set that we computed for the training set.

**Create Training and Test Sets and Apply Scaling**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

from sklearn.preprocessing import MinMaxScaler  
scaler = MinMaxScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)

**Build Models**

**Logistic Regression**

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function.

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression()  
logreg.fit(X\_train, y\_train)

print('Accuracy of Logistic regression classifier on training set: {:.2f}'  
 .format(logreg.score(X\_train, y\_train)))  
print('Accuracy of Logistic regression classifier on test set: {:.2f}'  
 .format(logreg.score(X\_test, y\_test)))

***Accuracy of Logistic regression classifier on training set: 0.70  
Accuracy of Logistic regression classifier on test set: 0.40***

Do you understand how does logistic regression work? If your answer is yes, I have a challenge for you to solve. Here is an extremely simple logistic problem.

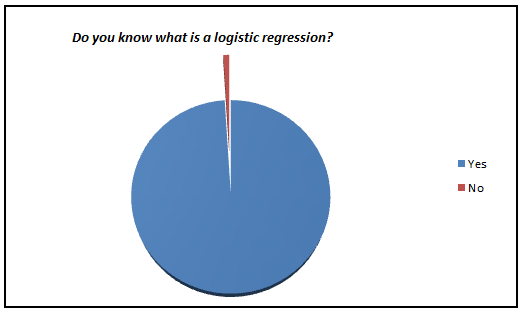
X = { 1,2,3,4,5,6,7,8,9,10}

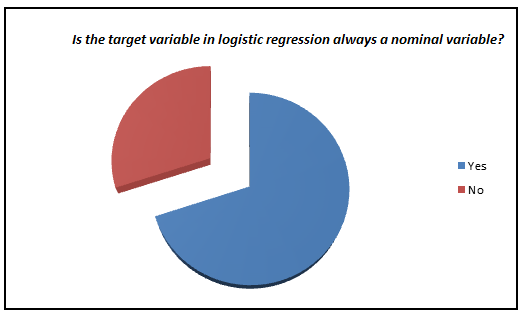
Y = {0,0,0,0,1,0,1,0,1,1}

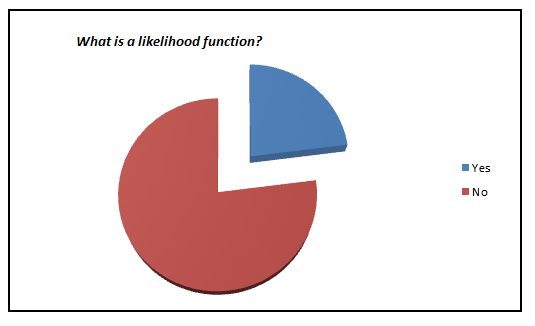
**Here is the catch : YOU CANNOT USE ANY PREDEFINED LOGISTIC FUNCTION!**

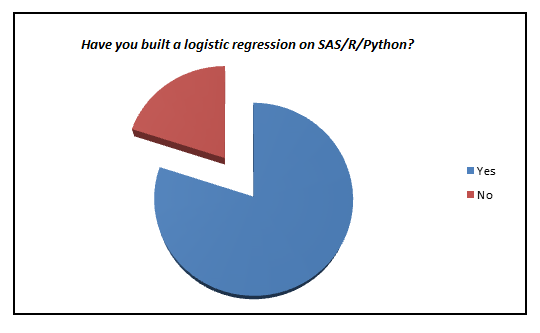
## Why am I asking you to build a Logistic Regression from scratch?

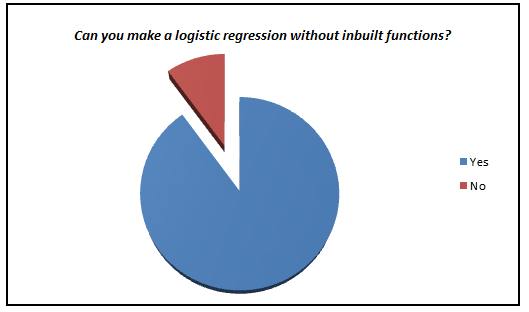
Here is a small survey which I did with professionals with 1-3 years of experience in analytics industry (my sample size is ~200).

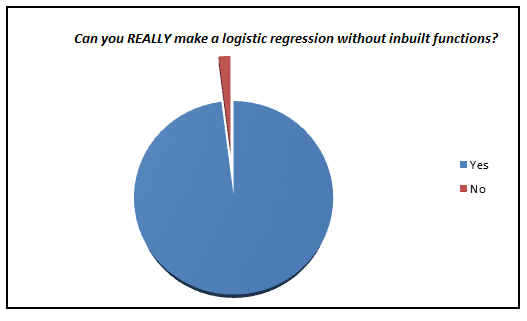








[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/q5.png)

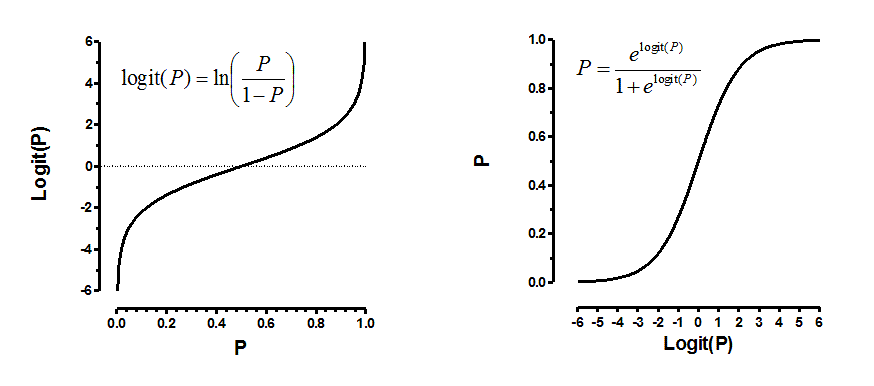


I was amazed to see such low percent of analyst who actually knows what goes behind the scene. We have now moved towards a generation where we are comfortable to see logistic regression also as a black box. In this article, I aim to kill this problem for once and all. The objective of the article is to bring out how logistic regression can be made without using inbuilt functions and not to give an introduction on Logistic regression.

## Refreshers of mathematics terminology

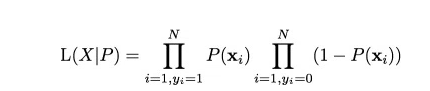
Logistic regression is an estimation of Logit function. Logit function is simply a log of odds in favor of the event. This function creates a s-shaped curve with the probability estimate, which is very similar to the required step wise function. Here goes the first definition :

### Logit Function:

Logistic regression is an estimate of a logit function. Here is how the logit function looks like:

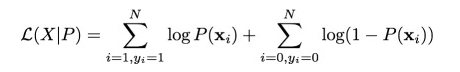
Now that you know what we are trying to estimate, next is the definition of the function we are trying to optimize to get the estimates of coefficient. This function is analogous to the square of error in linear regression and is known as the likelihood function. Here goes our next definition.

### Likelihood Function



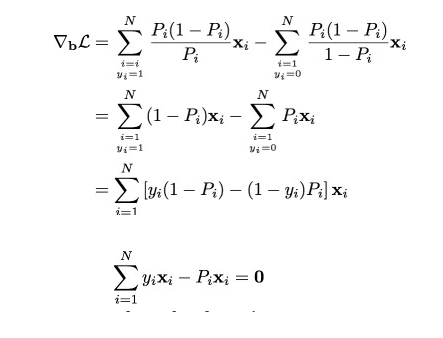
Given the complicated derivative of the likelihood function, we consider a monotonic function which can replicate the likelihood function and simplify derivative. This is the log of likelihood function. Here goes the next definition.

### Log Likelihood

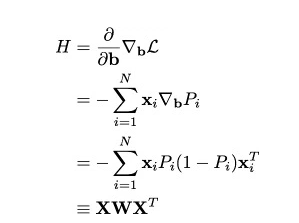


Finally we have the derivatives of log likelihood function. Following are the first and second derivative of log likelihood function.

### Derivative of Likelihood Function



### Hessian Matrix (second derivative)

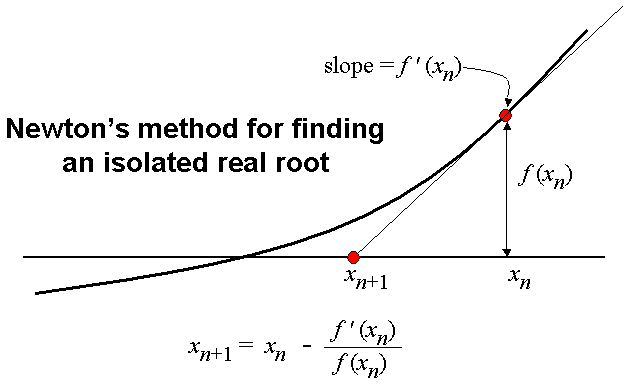


Finally, we are looking to solve the following equation.

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

As we now have all the derivative, we will finally apply the Newton Raphson method to converge to optimal solution. Here is a recap of Newton Raphson method.

### Newton Raphson****Method****



## The Code

Here is a R code which can help you make your own logistic function

Let’s get our functions right.

#Calculate the first derivative of likelihood function given output (y) , input (x) and pi (estimated probability)

calculateder <- function(y,x,pi) {

derv <- y\*x - pi\*x

derv\_sum <- sum(derv)

return(derv\_sum)

}

#Calculate the likelihood function given output(y) and pi

calculatell <- function(y,pi) {

ll <- 1

ll\_unit <- 1:length(y)

for (i in 1:length(y)){

ll\_unit[i] <- ifelse(y[i] == 1,pi[i],1-pi[i])

ll = ll\_unit[i]\*ll

}

return(ll)

}

#Calculate the value of pi (predictions on each observation) given x\_new(input) and estimated betas

findpi <- function(x\_new,beta){

pi <- 1:nrow(x\_new)

expon <- 1:nrow(x\_new)

for (i in 1:nrow(x\_new)){

expon[i] <- 0

for (j in 1:ncol(x\_new)){

expo <- x\_new[i,j] \* beta[j]

expon[i] <- expo + expon[i]}

pi[i] <- exp(expon[i])/(1+exp(expon[i]))

}

return(pi)

}

#Calculate the matrix W with all diagnol values as pi

findW <- function(pi){

W <- matrix(0,length(pi),length(pi))

for (i in 1:length(pi)){

W[i,i] <- pi[i]\*(1-pi[i])

}

return(W)

}

# Lets now make the logistic function given list of required inputs

logistic <- function(x,y,vars,obs,learningrate,dif) {

beta <- rep(0, (vars+1))

bias <- rep(1, obs)

x\_new <- cbind(bias,x)

derivative <- 1:(vars+1)

diff <- 10000

while(diff > dif) {

pi <- findpi(x\_new,beta)

pi <- as.vector(pi)

W <- findW(pi)

derivative <- (solve(t(x\_new)%\*%W%\*%as.matrix(x\_new))) %\*% (t(x\_new)%\*%(y - pi))

beta = beta + derivative

diff <- sum(derivative^2)

ll <- calculatell(y,pi)

print(ll)

}

return(beta)

}

# Time to test our algorithm with the values we mentioned at the start of the article

x <- 1:10

y <- c(rep(0, 4),1,0,1,0,1,1)

a <- logistic(x,y,1,10,0.01,0.000000001)

calculatell(y,findpi(x\_new,a))

#Log Likelihood = 0.01343191

data <- cbind(x,y)

data <- as.data.frame(data)

mylogit <- glm(y ~ x, data = data, family = "binomial")

mylogit

preds <- predict(mylogit, newdata = data,type ="response")

calculatell(data$y,preds)

#Log Likelihood = 0.01343191

#Isn't this amazing!!!

**Decision Tree**

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier().fit(X\_train, y\_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'  
 .format(clf.score(X\_train, y\_train)))  
print('Accuracy of Decision Tree classifier on test set: {:.2f}'  
 .format(clf.score(X\_test, y\_test)))

***Accuracy of Decision Tree classifier on training set: 1.00  
Accuracy of Decision Tree classifier on test set: 0.73***

**K-Nearest Neighbors**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()  
knn.fit(X\_train, y\_train)  
print('Accuracy of K-NN classifier on training set: {:.2f}'  
 .format(knn.score(X\_train, y\_train)))  
print('Accuracy of K-NN classifier on test set: {:.2f}'  
 .format(knn.score(X\_test, y\_test)))

***Accuracy of K-NN classifier on training set: 0.95  
Accuracy of K-NN classifier on test set: 1.00***

**Linear Discriminant Analysis**

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()  
lda.fit(X\_train, y\_train)  
print('Accuracy of LDA classifier on training set: {:.2f}'  
 .format(lda.score(X\_train, y\_train)))  
print('Accuracy of LDA classifier on test set: {:.2f}'  
 .format(lda.score(X\_test, y\_test)))

***Accuracy of LDA classifier on training set: 0.86  
Accuracy of LDA classifier on test set: 0.67***

**Gaussian Naive Bayes**

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()  
gnb.fit(X\_train, y\_train)  
print('Accuracy of GNB classifier on training set: {:.2f}'  
 .format(gnb.score(X\_train, y\_train)))  
print('Accuracy of GNB classifier on test set: {:.2f}'  
 .format(gnb.score(X\_test, y\_test)))

***Accuracy of GNB classifier on training set: 0.86  
Accuracy of GNB classifier on test set: 0.67***

**Support Vector Machine**

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. The model extracts a best possible hyper-plane / line that segregates the two classes.

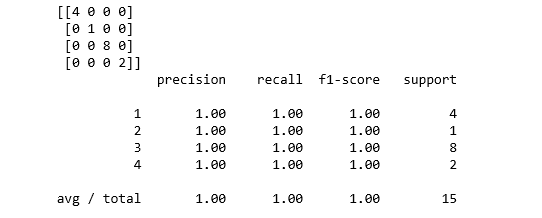
from sklearn.svm import SVC

svm = SVC()  
svm.fit(X\_train, y\_train)  
print('Accuracy of SVM classifier on training set: {:.2f}'  
 .format(svm.score(X\_train, y\_train)))  
print('Accuracy of SVM classifier on test set: {:.2f}'  
 .format(svm.score(X\_test, y\_test)))

***Accuracy of SVM classifier on training set: 0.61  
Accuracy of SVM classifier on test set: 0.33***

The KNN algorithm was the most accurate model that we tried. The confusion matrix provides an indication of no error made on the test set. However, the test set was very small.

from sklearn.metrics import classification\_report  
from sklearn.metrics import confusion\_matrix  
pred = knn.predict(X\_test)  
print(confusion\_matrix(y\_test, pred))  
print(classification\_report(y\_test, pred))



**Plot the Decision Boundary of the k-NN Classifier**

import matplotlib.cm as cm  
from matplotlib.colors import ListedColormap, BoundaryNorm  
import matplotlib.patches as mpatches  
import matplotlib.patches as mpatches

X = fruits[['mass', 'width', 'height', 'color\_score']]  
y = fruits['fruit\_label']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

def plot\_fruit\_knn(X, y, n\_neighbors, weights):  
 X\_mat = X[['height', 'width']].as\_matrix()  
 y\_mat = y.as\_matrix()

# Create color maps  
 cmap\_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF','#AFAFAF'])  
 cmap\_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF','#AFAFAF'])

clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights)  
 clf.fit(X\_mat, y\_mat)

# Plot the decision boundary by assigning a color in the color map  
 # to each mesh point.  
   
 mesh\_step\_size = .01 # step size in the mesh  
 plot\_symbol\_size = 50  
   
 x\_min, x\_max = X\_mat[:, 0].min() - 1, X\_mat[:, 0].max() + 1  
 y\_min, y\_max = X\_mat[:, 1].min() - 1, X\_mat[:, 1].max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, mesh\_step\_size),  
 np.arange(y\_min, y\_max, mesh\_step\_size))  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

# Put the result into a color plot  
 Z = Z.reshape(xx.shape)  
 plt.figure()  
 plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)

# Plot training points  
 plt.scatter(X\_mat[:, 0], X\_mat[:, 1], s=plot\_symbol\_size, c=y, cmap=cmap\_bold, edgecolor = 'black')  
 plt.xlim(xx.min(), xx.max())  
 plt.ylim(yy.min(), yy.max())

patch0 = mpatches.Patch(color='#FF0000', label='apple')  
 patch1 = mpatches.Patch(color='#00FF00', label='mandarin')  
 patch2 = mpatches.Patch(color='#0000FF', label='orange')  
 patch3 = mpatches.Patch(color='#AFAFAF', label='lemon')  
 plt.legend(handles=[patch0, patch1, patch2, patch3])

plt.xlabel('height (cm)')  
plt.ylabel('width (cm)')  
plt.title("4-Class classification (k = %i, weights = '%s')"  
 % (n\_neighbors, weights))   
plt.show()

plot\_fruit\_knn(X\_train, y\_train, 5, 'uniform')

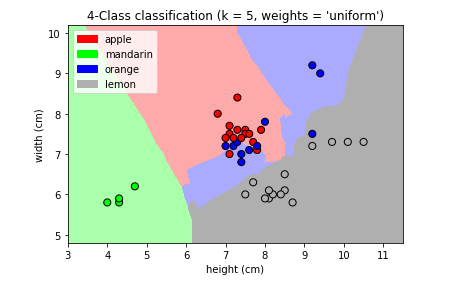


Figure 8

k\_range = range(1, 20)  
scores = []

for k in k\_range:  
 knn = KNeighborsClassifier(n\_neighbors = k)  
 knn.fit(X\_train, y\_train)  
 scores.append(knn.score(X\_test, y\_test))  
plt.figure()  
plt.xlabel('k')  
plt.ylabel('accuracy')  
plt.scatter(k\_range, scores)  
plt.xticks([0,5,10,15,20])

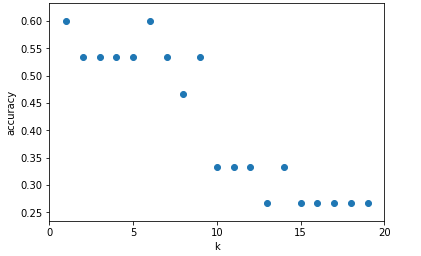


Figure 9

For this particular dateset, we obtain the highest accuracy when k=5.

**Summary**

In this post, we focused on the prediction accuracy. Our objective is to learn a model that has a good generalization performance. Such a model maximizes the prediction accuracy. We identified the machine learning algorithm that is best-suited for the problem at hand (i.e. fruit types classification); therefore, we compared different algorithms and selected the best-performing one

different classification algorithms

* Decision tree
* RandomForest - Ensemble method
* XGBoost
* SVM (Support Vector Machine) Classifier
* Nearest Neighbors Classifier
* SGD (Stochastic Gradient Descent) classifier
* Gaussian Naive Bayes
* MLP (Multi-layer Perceptron) Neural network

import numpy as np

import pandas as pd

*# ------ Load iris -----------*

iris = pd.read\_csv("../input/Iris.csv") *#load the dataset*

print(iris[0:2])

X = iris.iloc[:,1:5] *# ignore first column which is row Id*

y = iris.iloc[:,5:6] *# Classification on the 'Species'*

*# build train and test dataset*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.4, random\_state = 100)

from sklearn.metrics import accuracy\_score

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

fig = plt.figure(figsize=(15,20))

def run\_model(model, alg\_name, plot\_index):

*# build the model on training data*

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

*# make predictions for test data*

y\_pred = model.predict(X\_test)

*# calculate the accuracy score*

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

*# Compare the prediction result with ground truth*

color\_code = {'Iris-virginica':'red', 'Iris-setosa':'blue', 'Iris-versicolor':'green'}

*# plt.figure(plot\_index)*

ax = fig.add\_subplot(4,2,plot\_index)

colors = [color\_code[x] for x **in** y\_test.iloc[:,0]]

ax.scatter(X\_test.iloc[:,0], X\_test.iloc[:,3], color=colors, marker='.', label='Circle = Ground truth')

colors = [color\_code[x] for x **in** y\_pred]

ax.scatter(X\_test.iloc[:, 0], X\_test.iloc[:,3], color=colors, marker='o', facecolors='none', label='Dot = Prediction')

*#plt.axes([0.65, 0.65, 0.2, 0.2])*

ax.legend(loc="lower right")

*# manually set legend color to black*

leg = plt.gca().get\_legend()

leg.legendHandles[0].set\_color('black')

leg.legendHandles[1].set\_color('black')

leg.legendHandles[1].set\_facecolors('none')

ax.set\_title(alg\_name + ". Accuracy: " + str(accuracy))

*# ---- Decision Tree -----------*

from sklearn import tree

model = tree.DecisionTreeClassifier(criterion='entropy', max\_depth=5)

run\_model(model, "Decision Tree", 1)

*# ----- Random Forest ---------------*

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=10)

run\_model(model, "Random Forest", 2)

*# ----- xgboost ------------*

*# install xgboost*

*# 'pip install xgboost' or https://stackoverflow.com/questions/33749735/how-to-install-xgboost-package-in-python-windows-platform/39811079#39811079*

from xgboost import XGBClassifier

model = XGBClassifier()

run\_model(model, "XGBoost", 3)

*# ------ SVM Classifier ----------------*

from sklearn.svm import SVC

model = SVC()

run\_model(model, "SVM Classifier", 4)

*# -------- Nearest Neighbors ----------*

from sklearn import neighbors

model = neighbors.KNeighborsClassifier()

run\_model(model, "Nearest Neighbors Classifier", 5)

*# ---------- SGD Classifier -----------------*

from sklearn.linear\_model import SGDClassifier

from sklearn.multiclass import OneVsRestClassifier

model = OneVsRestClassifier(SGDClassifier())

run\_model(model, "SGD Classifier", 6)

*# --------- Gaussian Naive Bayes ---------*

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

run\_model(model, "Gaussian Naive Bayes", 7)

*# ----------- Neural network - Multi-layer Perceptron ------------*

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier()

run\_model(model, " MLP Neural network ", 8)