Chapter 1: Building a AI Model to detect Breast Cancer Prediction

Introduction: We will Build a Breast Cancer Prediction Model with Machine Learning Algorithms (Random Forest, Ensemble Classifier etc ) & Deep Learning Framework with Tensorflow.

**Structure:**

* Problem Statement
* [Load Python libraries and importing the data](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "1)
* [Exploratory & Statistical Data Analysis (EDA)](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "2)
* [Principal Component Analysis](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "3)
* [Automating functions for evaluation](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "4)
* [Preparation of dataset](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "5) for Machine Learning
* [Predictive model 1: Random Forest Regression](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "6)
* [Predictive model 2: Logistic Regression](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "7)
* [Predictive model 3:Ensemble Classifier to maximise precision &detect all malignant tumors](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "8)
* [Predictive model 4 : Deep Learning ANN Model using Tensorflow to predict](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "9)
* Summary

**Objectives**

We will cover the following recipes:

* Various Statistical Analysis & Visualization using one of the best tool (Plotly)
* Preprocessing data using different technique &Feature Extraction.
* Finding optimal hyperparameters 7 Tackling class imbalance.
* Evaluating the performance of Classification algorithms.
* Working on Different Deep Learning & Machine Learning Hybrid Models.
* Understand the TensorFlow & Keras ecosystem using various datasets and techniques complete end to end Framework of Artificial Intelligence.

**1.1 Problem Statement**

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

Also can be found on UCI Machine Learning Repository: [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

Attribute Information:

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

**1.2 [Load Python libraries and importing the data](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "1)**

**1.2.1 [Loading of Dependent libraries](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "1.1)**

* from sklearn.decomposition import PCA
* from sklearn.linear\_model import LogisticRegression
* from sklearn.ensemble import VotingClassifier
* import plotly.offline as py

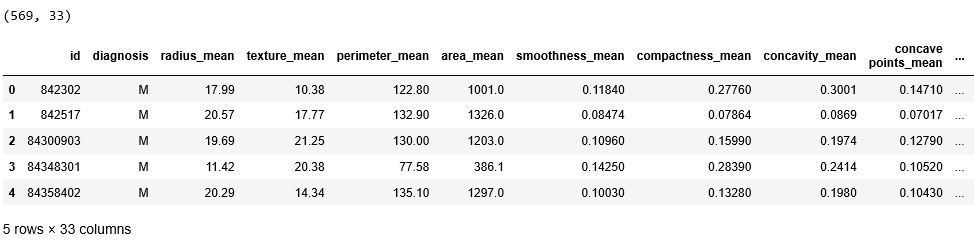
|  |
| --- |
| Rest all the imports I have showed in my Jupyter Notebook, which I gave hyperlink of my Github Account of this chapter.Note:plotly==4.0.0 plotly-express==0.4.1 Anaconda Package Python 2.x/3.x, TensorFlow, Keras.  **Note**  Code Repository: <https://github.com/aniruddhachoudhury/Artificial-Intelligence-Projects-/tree/master/Chapter1> |

* import tensorflow as tf

**1.2.2 [Reading the data from Local File](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "1.2)**

data = pd.read\_csv('data.csv')

print(data.shape)

**Table:** *Table 1.1:* *Breast Cancer Data-set*

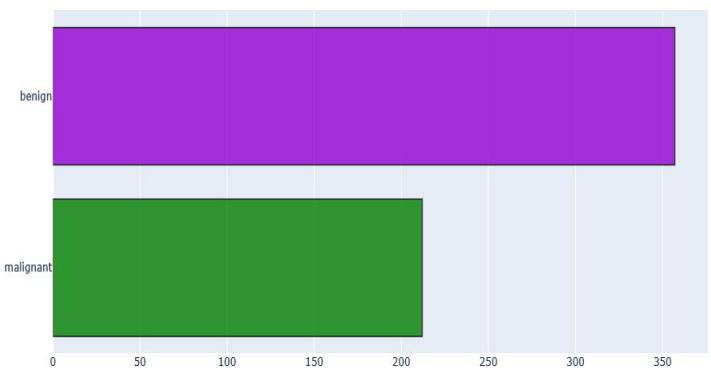
We Loaded the Data-set & here we having 569 rows & 33 columns as features.

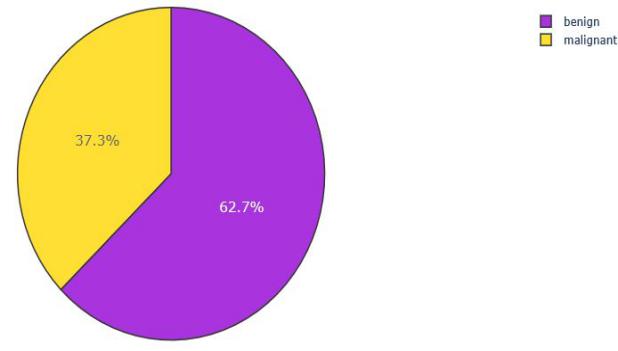
**1.3.[Exploratory & Statistical Data Analysis (EDA)](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "2)**

We often jump from problem-to-problem in applied machine learning and we need to get up to speed on a new dataset, fast.A classical and under-utilised approach that you can use to quickly build a relationship with a new data problem is ****Exploratory Data Analysis****

**1.3.1 Feature Target distribution (number and %)**

Malignant= data[(data['diagnosis'] == 1)]

Benign = data[(data['diagnosis'] == 0)]

****

*Figure 1.1: Figure 1.2:*

As we can see here we have 37.3% of Malignant & 62.7% of Benign distribution across Data.

**1.3.2 Features distribution across (hue = diagnosis)**

How many times each value appears in data set. This description is called the distribution of variable Most common way to represent distribution of variable is histogram that is graph which shows frequency of each value. Frequency = number of times each value appears

def **plot\_distribution**(data\_select, size\_bin):

temp1 = Malignant[data\_select]

temp2 = Benign[data\_select]

hist\_data = [temp1, temp2]

group\_labels = ['malignant', 'benign'] # Category Distribution

colors = ['#ff7f0e', '#bcbd22']

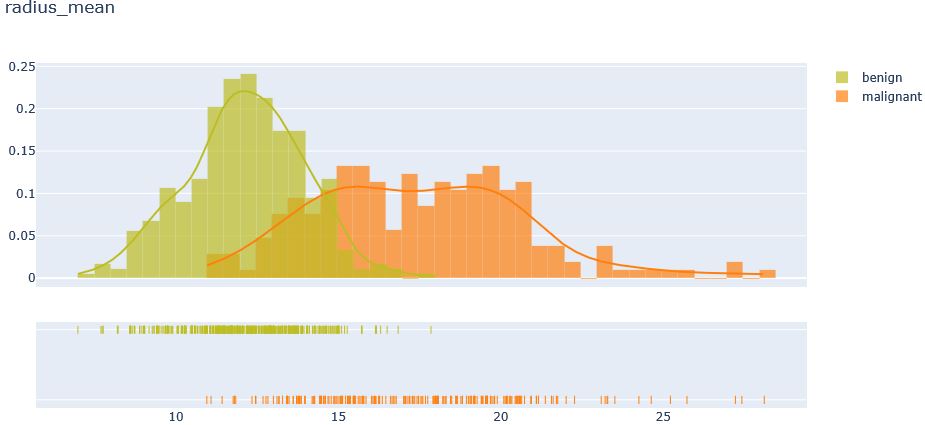
fig = ff.create\_distplot(hist\_data, group\_labels, colors = colors, show\_hist = True,bin\_size = size\_bin, curve\_type='kde')

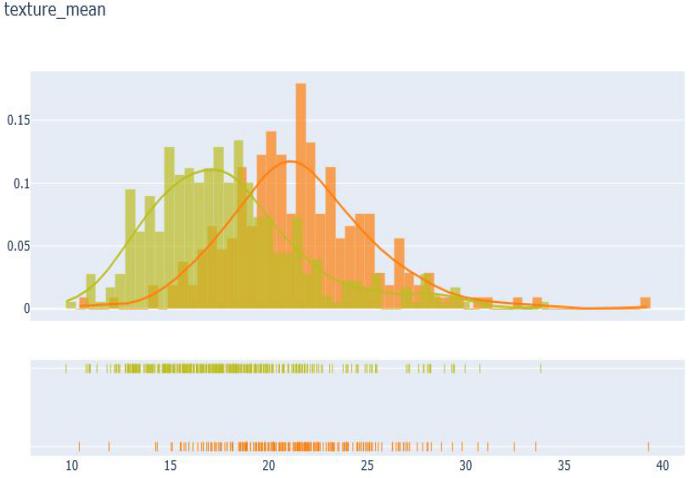
fig['layout'].update(title = data\_select)

py.iplot(fig, filename = 'Density plot')

return

plot\_distribution('radius\_mean', .5) plot\_distribution('texture\_mean', .5)



*Figure 1.3 Figure 1.4:* ****

As we have plotted this using Plotly and we have found in above pics that the distribution we created with category Malignant & Benign a histogram distribution with Normal Distribution Curve with .5 binning Size for Radius Mean & Texture Mean, we founded that in texture mean both category distributed equal and in Radius Mean the normal distribution curve of Benign is more than Malignant and it is towards Negative Skewed. Similarly we plotted for other attributes.

**1.3.3 Correlation matrix between Features**

A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable’s value increases, the other variables’ values decrease. Correlation can also be neural or zero, meaning that the variables are unrelated.

* ****Positive Correlation****: both variables change in the same direction.
* ****Neutral Correlation****: No relationship in the change of the variables.
* ****Negative Correlation****: variables change in opposite directions.

The performance of some algorithms can deteriorate if two or more variables are tightly related, called multicollinearity.

correlation = data.corr()

#tick labels

matrix\_cols = correlation.columns.tolist()

#convert to array

corr\_array = np.array(correlation)

#Plotting

trace = go.Heatmap(z = corr\_array,x = matrix\_cols,y = matrix\_cols,xgap = 2,ygap = 2,

colorscale='Viridis',colorbar = dict())

layout = go.Layout(dict(title = 'Correlation Matrix for variables', autosize = False,

height = 720,width = 800,margin = dict(r = 0 ,l = 210, t = 25,b = 210),

yaxis = dict(tickfont = dict(size = 9)),

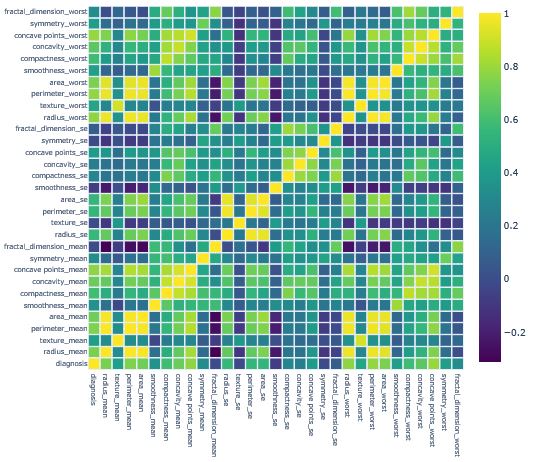
xaxis = dict(tickfont = dict(size = 9)))

)

fig = go.Figure(data = [trace],layout = layout)

py.iplot(fig)

*Figure 1.5*

Here we created Heat Map by passing the Correlation Array Matrix and made a setting for Pixel, Colour, Range in plotly Huge matrix that includes a lot of numbers The range of this numbers are -1 to 1. Meaning of 1 is two variable are positively correlated with each other like radius mean and area mean Meaning of zero is there is no correlation between variables like radius mean and fractal dimension .Meaning of -1 is two variables are negatively correlated with each other like radius mean and fractal dimension mean.Actually correlation between of them is not -1, it is -0.3 but the idea is that if sign of correlation is negative that means that there is negative correlation.

**1.3.4 [Correlated features with Positive](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "2.6) / Negative**

**def plot\_attributes\_Correlation(att1, att2) :**

**trace1 = go.Scatter(x = Malignant[att1],y = Malignant[att2],name = 'Malignant',mode = 'markers', marker = dict(color = '#d62728',line = dict(width = 1)))**

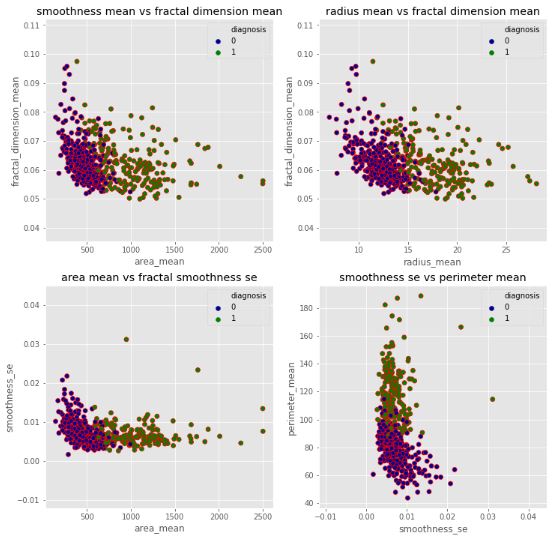
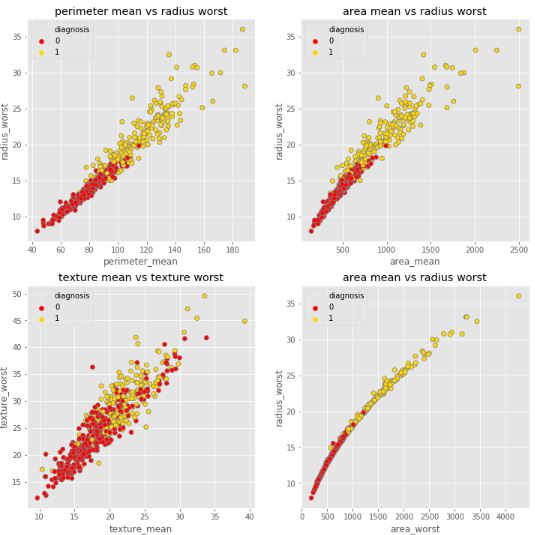
**trace2 = go.Scatter(x = Benign[att1],y = Benign[att2],name = 'Benign',mode = 'markers', marker = dict(color = '#7EC0EE',line = dict(width = 1)))**

**layout = dict(title = att1 +" "+"vs"+" "+ att2,yaxis = dict(title = att2,zeroline = False),**

**xaxis dict(title= att1, zeroline = False))**

**plots = [trace1, trace2] fig = dict(data = plots,layout=layout)**

**py.iplot(fig)**

*Figure 1.******6 Figure 1.7*

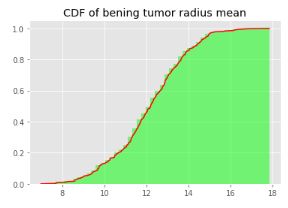
As we can see Figure 1.6 is positive correlation between

('texture\_mean','texture\_worst')('area\_mean','radius\_worst')('perimeter\_mean','radius\_worst')('area\_worst','radius\_worst')

And Figure 1.7 negative correlation between

('area\_mean','smoothness\_se')('smoothness\_se','perimeter\_mean')('area\_mean','fractal\_dimension\_mean')('radius\_mean','fractal\_dimension\_mean')

**1.3.5 Cumulative distribution function**

CDF is the probability that the variable takes a value less than or equal to x. P(X <= x)Lets explain in cdf graph of bening radius mean in graph, what is P(12 < X)?

**plt.hist(data\_bening.radius\_mean,bins=50,fc=(0,1,0,0.5),label='Bening',normed = True,cumulative = True)**

**sorted\_data = np.sort(data\_bening.radius\_mean)**

**y = np.arange(len(sorted\_data))/float(len(sorted\_data)-1)**

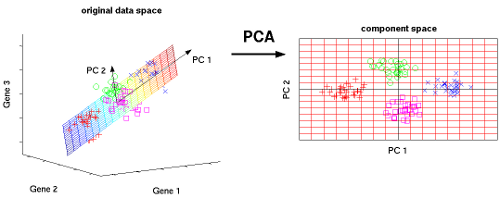
**plt.plot(sorted\_data,y,color='red') plt.show()** *Figure 1.8*

**1.4 [Principal Component Analysis](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "3)**

So PCA is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data.

It is always performed on a symmetric correlation or covariance matrix. This means the matrix should be numeric and have standardized data.Summarizing it, given the original feature space, PCA finds a linear projection of itself in a lower dimensional space that has the following properties: The conserved variance is maximized. The final reconstruction error (when trying to go back from transformed features to original ones) is minimized.

The image below shows the transformation of a high dimensional data (3 dimension) to low dimensional data (2 dimension) using PCA. Not to forget, each resultant dimension is a linear combination of *p* features

****

*Figure 1.9*

A principal component is a normalized linear combination of the original predictors in a data set. In image above, *PC1* and *PC2* are the principal components. Let’s say we have a set of predictors as X¹, X²...,Xp

The principal component can be written as: Z¹ = Φ¹¹X¹ + Φ²¹X² + Φ³¹X³ + .... +Φp¹Xp

where, Z¹ is first principal component, Φp¹ is the loading vector comprising of loadings (Φ¹, Φ²..) of first principal component. Closeness is measured using average squared euclidean distance.X¹..Xp are normalized predictors. Normalized predictors have mean equals to zero and standard deviation equals to one.

**1.4.1 Compute PCA reducing Technique with 6components** target\_pca = data['diagnosis']

data\_pca = data.drop('diagnosis', axis=1)

target\_pca = pd.DataFrame(target\_pca)

#To make a PCA, normalize data is essential for bringing in one scale to reduce variance

X\_pca = data\_pca.values

X\_std = StandardScaler().fit\_transform(X\_pca)

pca = PCA(svd\_solver='full')

pca\_std = pca.fit(X\_std, target\_pca).transform(X\_std)

pca\_std = pd.DataFrame(pca\_std)

pca\_std = pca\_std.merge(target\_pca, left\_index = True, right\_index = True, how = 'left')

pca\_std['diagnosis'] = pca\_std['diagnosis'].replace({1:'malignant',0:'benign'})

#explained\_variance in Dataframe

var\_pca = pd.DataFrame(pca.explained\_variance\_ratio\_)

var\_pca = var\_pca.T

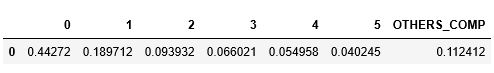
#----------SUM AND DROP COMP [7:30]

col\_list = list(v for v in chain(pca\_std.columns[6:30]))

var\_pca['OTHERS\_COMP'] = var\_pca[col\_list].sum(axis=1)

var\_pca.drop(var\_pca[col\_list],axis=1,inplace=True)

var\_pca = var\_pca.T



*Table 1.2: Explained Variance Ratio*

def plot\_PCA():

labels = ['ATT1','ATT2','ATT3','ATT4','ATT5','ATT6', 'ATT7 - 30']

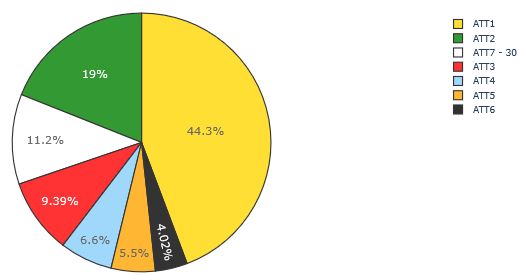
colors = ['gold', 'green', 'red', 'lightskyblue', 'orange', 'black', 'white']

trace = go.Pie(labels = labels, values = var\_pca[0].values, opacity = 0.8,

textfont=dict(size=15), marker=dict(colors=colors, line=dict(color='#000000', width=1.5)))

layout = dict(title = 'PCA : components and explained variance (6 comp = 88.8%)')

fig = dict(data = [trace], layout=layout)

py.iplot(fig)

return

plot\_PCA()

As we have selected first 6 components which have the Most number of Variance across all the components rest we have excluded. And we plotted the variance in Plotly Pie chart Distribution

*Figure 1.10*

**1.5 Automating functions for evaluation**

**1.5.1 Confusion matrix and show metrics**

The [confusion matrix](https://en.wikipedia.org/wiki/Confusion_matrix" \t "http://localhost:8888/notebooks/Downloads/Book%20BPP/Breast%20Cancer/_blank), also known as the error matrix, allows visualization of the performance of an algorithm :

* true positive (TP) : Malignant tumour correctly identified as malignant
* true negative (TN) : Benign tumour correctly identified as benign
* false positive (FP) : Benign tumour incorrectly identified as malignant
* false negative (FN) : Malignant tumour incorrectly identified as benign

Metrics :

* Accuracy : (TP +TN) / (TP + TN + FP +FN)
* Precision : TP / (TP + FP)
* Recall : TP / (TP + FN)

def plot\_confusion\_matrix(cm, classes,normalize = False,title = 'Confusion matrix"',

cmap = plt.cm.Blues):

plt.imshow(cm, interpolation = 'nearest', cmap = cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation = 0)

plt.yticks(tick\_marks, classes)

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])) :

plt.text(j, i, cm[i, j],

horizontalalignment = 'center',color = 'white' if cm[i, j] > thresh else 'black')

plt.tight\_layout()

plt.ylabel('True label')

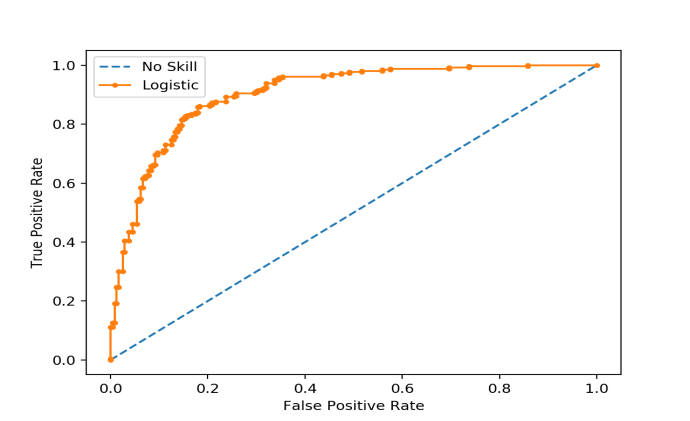
plt.xlabel('Predicted label')

return

**1.5.2 Learning curve**

The precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall)

def plot\_precision\_recall():

plt.step(recall, precision, color = 'b',

alpha = 0.2,where = 'post')

plt.fill\_between(recall, precision, step

='post’, alpha = 0.2, color = 'b')

plt.plot(recall, precision, linewidth=2)

plt.xlim([0.0,1])

plt.ylim([0.0,1.05])

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision Recall Curve')

plt.show();

*Figure 1.11*

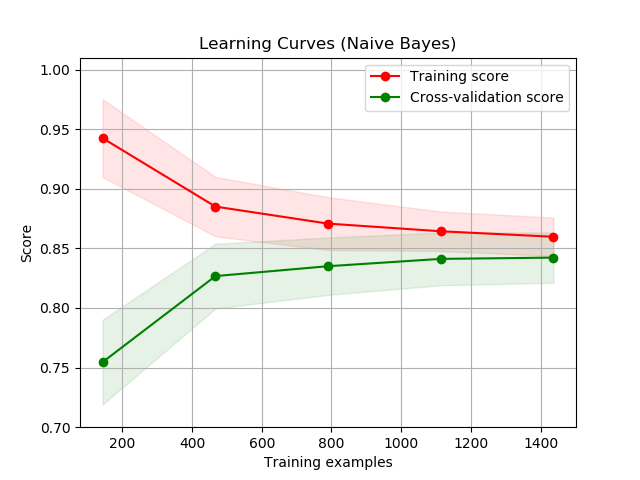
**1.5.3 Learning curve**

It usually refers to a plot of the *prediction accuracy/error* vs. the *training set size* (i.e: how better does the model get at predicting the target as you the increase number of instances used to train it)

Usually both the training and test/validation performance are plotted together so we can diagnose the bias-variance tradeoff

def plot\_learning\_curve(estimator, title, X, y, ylim = None, cv = None,

n\_jobs = 1, train\_sizes = np.linspace(.1, 1.0, 5)):

 plt.figure()

plt.title(title)

if ylim is not None:

plt.ylim(\*ylim)

plt.xlabel('Training examples')

plt.ylabel('Score')

train\_sizes, train\_scores,test\_scores = learning\_curve(estimator, X, y, cv = cv,

n\_jobs = n\_jobs, train\_sizes = train\_sizes)

train\_scores\_mean = np.mean(train\_scores, axis = 1)

train\_scores\_std = np.std(train\_scores, axis = 1)

test\_scores\_mean = np.mean(test\_scores, axis = 1) *Figure 1.12*

test\_scores\_std = np.std(test\_scores, axis = 1)

plt.grid()

plt.fill\_between(train\_sizes,train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, alpha=0.1,color="r")

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, alpha = 0.1, color = "g")

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color = "r",label = "Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color = "g",label = "Cross-validation score")

plt.legend(loc = "best")

return plt

**1.5.4 ROC curve**

The [ROC curve](https://en.wikipedia.org/wiki/Receiver_operating_characteristic" \t "http://localhost:8888/notebooks/Downloads/Book%20BPP/Breast%20Cancer/_blank) is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings

def plot\_roc(fpr, tpr, label):

plt.plot(fpr, tpr, label = 'ROC curve', linewidth = 2)

plt.plot([0,1],[0,1], 'k--', linewidth = 2)

# plt.xlim([0.0,0.001])

# plt.ylim([0.0,1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.show()

return

**1.5.5 Cross validation metrics**

It a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it

def cross\_val\_metrics(model) :

scores = ['accuracy', 'precision', 'recall']

for sc in scores:

scores = cross\_val\_score(model, X, y, cv = 5, scoring = sc)

print('[%s] : %0.5f (+/- %0.5f)'%(sc, scores.mean(), scores.std()))

**1.6 Preparation of data-set for Machine Learning**

1>Defining Dataset

y = np.array(data.diagnosis.tolist())

data1=data

data1 = data1.drop(['diagnosis'], axis=1)

X = np.array(data1.as\_matrix())

2>Normalization

scaler = StandardScaler()

X = scaler.fit\_transform(X)

Why do you need to standardize your data ? For example, a variable that ranges between 0 and 100 will outweigh a variable that ranges between 0 and 1. Using these variables without standardization in effect gives the variable with the larger range a bigger weight in the analysis

3>Train\_test split using Sckit Learn

random\_state = 42

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = random\_state)

**Training Dataset**: The sample of data used to fit the model.

**Test Dataset**: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

**1.7 Feature Selection**

**1.7.1 Tree based feature selection and random forest classification**

In [random forest classification](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html" \t "http://localhost:8888/notebooks/Downloads/Book%20BPP/Breast%20Cancer/_blank) method there is a **feature\_importances\_** attributes that is the feature importances (the higher, the more important the feature). **!!! To use feature\_importance method, in training data there should not be correlated features. Random forest choose randomly at each iteration, therefore sequence of feature importance list can change.**

y1 = data['diagnosis']

X1 = data.drop(['diagnosis'], axis=1)

# Train\_test split

random\_state = 42

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X1, y1, test\_size = 0.30, random\_state = random\_state)

clf\_rf\_5 = RandomForestClassifier()

clr\_rf\_5 = clf\_rf\_5.fit(X\_train1,y\_train1)

importances = clr\_rf\_5.feature\_importances\_

std = np.std([tree.feature\_importances\_ for tree in clf\_rf.estimators\_],

axis=0)

indices = np.argsort(importances)[::-1]

# Print the feature ranking

print("Feature ranking:")

for f in range(X\_train1.shape[1]):

print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest

plt.figure(1, figsize=(14, 13))

plt.title("Feature importances")

plt.bar(range(X\_train1.shape[1]), importances[indices],

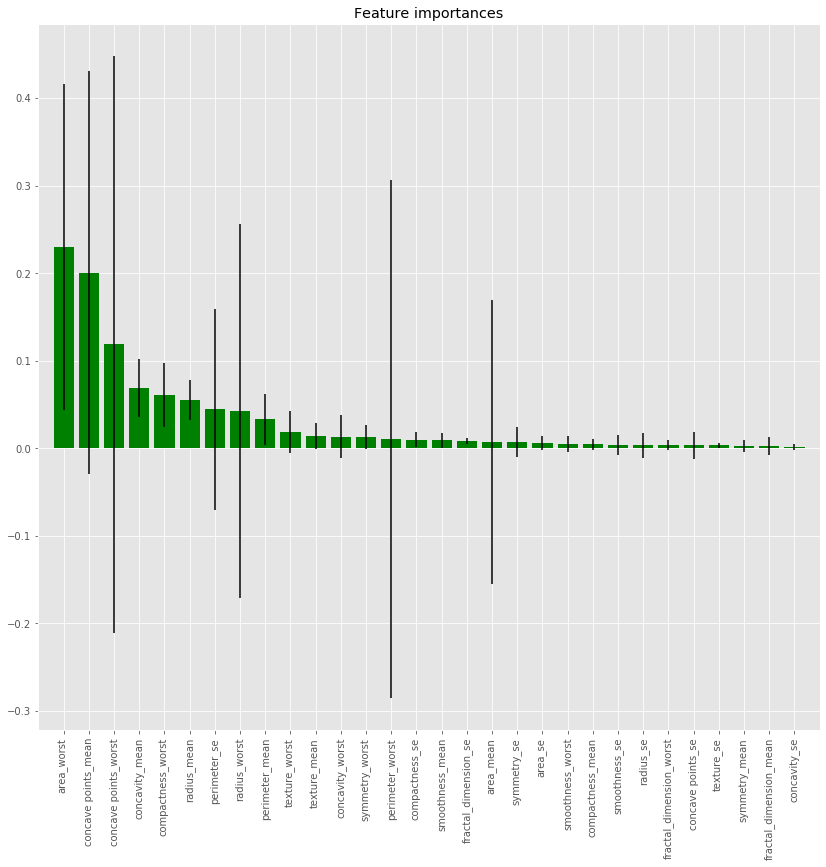
color="g", yerr=std[indices], align="center")

plt.xticks(range(X\_train1.shape[1]),

X\_train1.columns[indices],rotation=90)

plt.xlim([-1, X\_train1.shape[1]])

plt.show()

*Figure 1.13*

Here in above method clearly showing the important features as area\_worst,concave points\_mean, concave\_points\_worst,concavity\_mean.

**1.7.2 Univariate feature selection with random forest classification**

Basically, it uses one of the classification methods ([random forest](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html" \t "http://localhost:8888/notebooks/Downloads/Book%20BPP/Breast%20Cancer/_blank) in our example), assign weights to each of features. Whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

# find best scored 5 features

select\_feature = SelectKBest(chi2, k=6).fit(X\_train1, y\_train1)

dfscores = pd.DataFrame(select\_feature.scores\_)

dfcolumns = pd.DataFrame(X\_train1.columns)

#concat two dataframes for better visualization

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['Specs','Score'] #naming the dataframe columns

print(featureScores.nlargest(5,'Score'))

Capture

Chosen 5 best features by Univarate feature selection is ‘**area**\_mean', 'area\_worst’,’area\_se’,’perimeter\_worst’,’perimeter\_mean’****.Therefore we do not need to calculate accuracy again. Shortly, we can say that we make good feature selection with Ufs methods

**1.8. Predictive Model 1: Random forest classification**

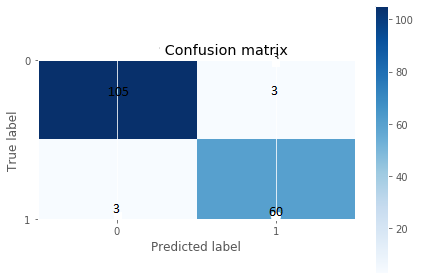
## IMG_256**How does the algorithm work?**

It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

x\_train\_2 = select\_feature.transform(X\_train1)

x\_test\_2 = select\_feature.transform(X\_test1) *Figure 1.14*



#random forest classifier with n\_estimators=10 (default)

clf\_rf\_2 = RandomForestClassifier()

clr\_rf\_2 = clf\_rf\_2.fit(x\_train\_2,y\_train)

y\_pred = clr\_rf\_2.predict(x\_test\_2)

# Confusion maxtrix & metrics

cm = confusion\_matrix(y\_test, y\_pred)

class\_names = [0,1]

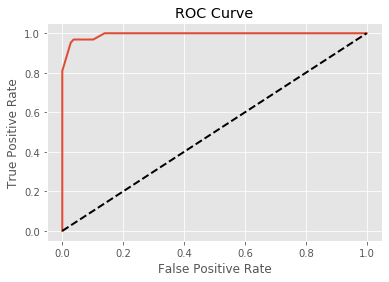
plt.figure()

plot\_confusion\_matrix(cm,classes=class\_names, *Figure 1.15*

Capturetitle='Logistic Confusion matrix')

show\_metrics()

*Figure 1.16*

# Probabilities for each class

clr\_probs = clr\_rf\_2.predict\_proba(x\_test\_2)[:, 1]

# ROC curve

fpr, tpr, t = roc\_curve(y\_test, clr\_probs)

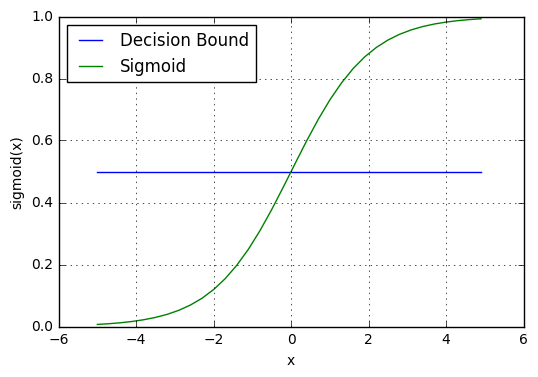
plot\_roc(fpr, tpr, t)

*Figure 1.17*

Accuracy is almost 96% and as it can be seen in confusion matrix, we make few wrong prediction. What we did up to now is that we choose features according to correlation matrix and according to selectkBest method. Although we use 5 features in selectkBest method accuracies look similar. Now lets see other feature selection methods to find better results.

**1.8. Predictive Model 2: Logistic Regression**

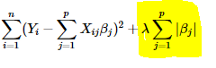
Logistic regression predictions are discrete (only specific values or categories are allowed). We can also view probability scores underlying the model’s classifications.Our current prediction function returns a probability score between 0 and 1

**Ridge regression** adds “*squared magnitude*” of coefficient as penalty term to the loss function. Here the *highlighted* part represents L2 regularization element.

IMG_257Cost function

Here, if *lambda* is zero then you can imagine we get back OLS. However, if *lambda* is very large then it will add too much weight and it will lead to under-fitting. Having said that it’s important how *lambda* is chosen. This technique works very well to avoid over-fitting issue. *Figure 1.18*

**Lasso Regression** (Least Absolute Shrinkage and Selection Operator) adds “*absolute value of magnitude*” of coefficient as penalty term to the loss function.

Cost function

Again, if *lambda* is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.The **key difference** between these techniques is that Lasso shrinks the less important feature’s coefficient to zero thus, removing some feature altogether. So, this works well for **feature selection** in case we have a huge number of features.

We will find the Best Parameters to fit the Model for Logistic Regression by Hyperparameter tuning for Best accuracy.

# Find best hyperparameters (accuracy)

log\_clf = LogisticRegression(random\_state = random\_state)

param\_grid = {

'penalty' : ['l2','l1'],

'C' : [0.001, 0.01, 0.1, 1, 10, 100, 1000]

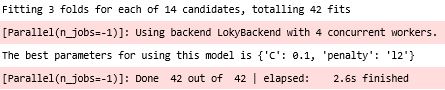
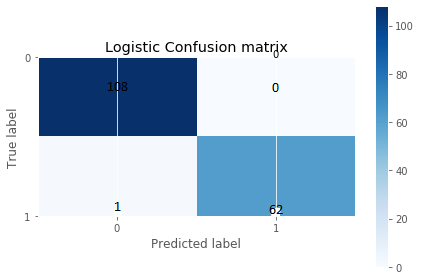
}

CV\_log\_clf = GridSearchCV(estimator = log\_clf, param\_grid = param\_grid , scoring = 'accuracy', verbose = 1, n\_jobs = -1)

CV\_log\_clf.fit(X\_train, y\_train)

best\_parameters = CV\_log\_clf.best\_params\_

print('The best parameters for using this model is', best\_parameters)



#Log with best hyperparameters

CV\_log\_clf = LogisticRegression(C = best\_parameters['C'],

penalty = best\_parameters['penalty'],

random\_state = random\_state)

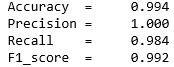
CV\_log\_clf.fit(X\_train, y\_train)

y\_pred = CV\_log\_clf.predict(X\_test)

y\_score = CV\_log\_clf.decision\_function(X\_test)

*Figure 1.19*

# Confusion maxtrix & metrics

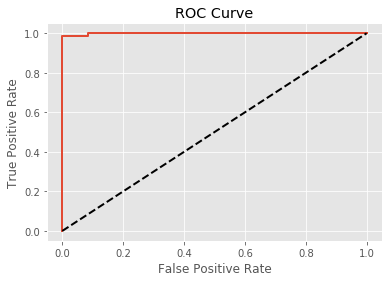
cm = confusion\_matrix(y\_test, y\_pred)

class\_names = [0,1]

plt.figure()

plot\_confusion\_matrix(cm,

classes=class\_names,title='Logistic Confusion matrix') *Figure 1.20*

**plt.show()

show\_metrics()

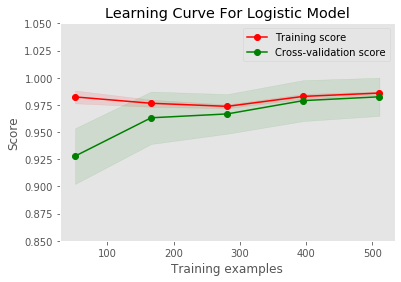
# ROC curve

fpr, tpr, t = roc\_curve(y\_test, y\_score)

plot\_roc(fpr, tpr, t)

*Figure 1.20*

**1.8.1 Validating learning curves and cross validation score**

****#Learning curve Log with best hyperparameter

plot\_learning\_curve(CV\_log\_clf,

'Learning Curve For Logistic Model',

X, y, (0.85,1.05), 10)

plt.show()

# Cross val Log

cross\_log = cross\_val\_metrics(CV\_log\_clf)

Capture

*Figure 1.21 Figure 1.22*

we got an accuracy of 98.24 with a standard deviation of 0.56 which is great accuracy and as we can see for a good fit is the goal of the learning algorithm and exists between an overfit and underfit model.

A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

The loss of the model will almost always be lower on the training dataset than the validation dataset. This means that we should expect some gap between the train and validation loss learning curves. This gap is referred to as the “generalization gap.”

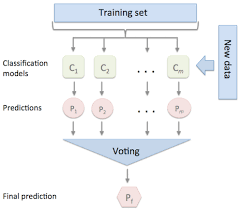
A plot of learning curves shows a good fit if:

* The plot of training loss decreases to a point of stability.
* The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

Continued training of a good fit will likely lead to an overfit.

The example plot below above demonstrates this case of a good fit.

**1.9 Predictive Model 3: Ensemble Classifier to maximise precision and detect all malignant tumors**

A collection of several models working together on a single set is called an ensemble. The method is called Ensemble Learning.

Voting is one of the simplest way of combining the predictions from multiple machine learning algorithms. Voting classifier isn’t an actual classifier but a wrapper for set of different ones that are trained and valuated in parallel in order to exploit the different peculiarities of each algorithm.

*Figure 1.23*

Again we created anothe Logistic Regression Model with Different set of Grid Search Hyperparemters.

# Find the best parameters (recall)

log2\_clf = LogisticRegression(random\_state = random\_state)

param\_grid = {

'penalty' : ['l2','l1'],

'C' : [0.001, 0.01, 0.1, 1, 10, 100, 1000],

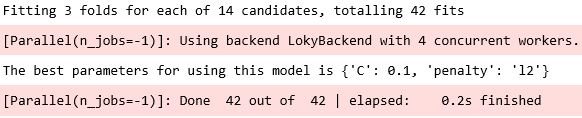
}

CV\_log2\_clf = GridSearchCV(estimator = log2\_clf, param\_grid = param\_grid , scoring = 'recall', verbose = 1, n\_jobs = -1)

CV\_log2\_clf.fit(X\_train, y\_train)

best\_parameters = CV\_log2\_clf.best\_params\_

print('The best parameters for using this model is', best\_parameters)



# Log w best hyperparameters

CV\_log2\_clf = LogisticRegression(C = best\_parameters['C'],penalty best\_parameters['penalty'], random\_state = random\_state)

CV\_log2\_clf.fit(X\_train, y\_train)

y\_pred = CV\_log2\_clf.predict(X\_test)

y\_score = CV\_log2\_clf.decision\_function(X\_test)

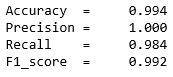
**1.9.1 Voting classifier : logistic1 + logistics2**

#Voting Classifier

voting\_clf = VotingClassifier (

estimators = [('log1', CV\_log\_clf), ('log\_2', CV\_log2\_clf)],

voting='soft', weights = [1, 1])

****

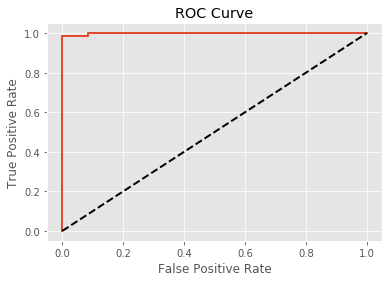
voting\_clf.fit(X\_train,y\_train)

y\_pred = voting\_clf.predict(X\_test)

y\_score = voting\_clf.predict\_proba(X\_test)[:,1]

# Confusion maxtrix *Figure 1.24*

cm = confusion\_matrix(y\_test, y\_pred)

class\_names = [0,1]

show\_metrics()

# ROC curve

fpr, tpr, t = roc\_curve(y\_test, y\_score)

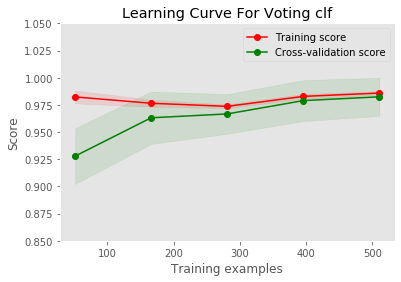
plot\_roc(fpr, tpr, t)

# Cross val score voting

cross\_voting = cross\_val\_metrics(voting\_clf)

Capture

*Figure 1.24 Figure 1.25*

**#Learning curve Voting

plot\_learning\_curve(voting\_clf,

'Learning Curve For Voting clf',

X, y, (0.85,1.05), 10)

plt.show() *Figure 1.25*

Here we used **Soft voting** that returns the class label as argmax of the sum of predicted probabilities.Specific weights can be assigned to each classifier via the weights parameter. When weights are provided, the predicted class probabilities for each classifier are collected, multiplied by the classifier weight, and averaged. The final class label is then derived from the class label with the highest average probability.And from the learning curve it shows the that model is a good fit with less variance and Bias, and we got a accuracy of 98% and standard deviation of 0.56.

**1.9.2 Models performance plot (accuracy, precision, recall)**

models\_score = {'Random\_Forest': [0.965, 0.952, 0.952],'log\_clf': [0.984, 0.990, 0.962],

'log2\_clf' : [0.974,0.976,0.953], 'voting\_clf' : [0.994,0.99,0.98]}

Performance = pd.DataFrame(data = models\_score)

Performance.rename(index={0:'Accuracy',1:'Precision', 2: 'Recall'},inplace=True)

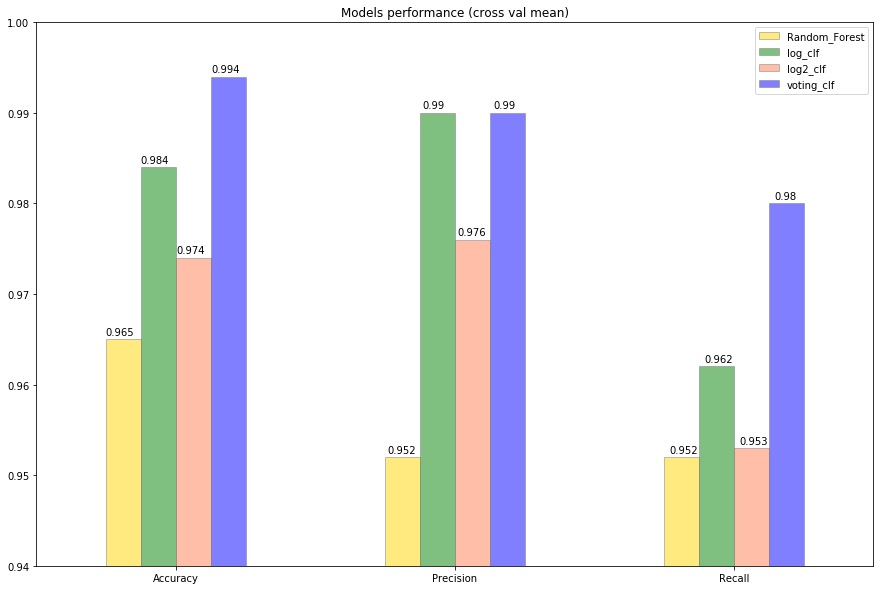
ax = Performance.plot(kind='bar', figsize = (15,10), ylim = (0.94, 1),

color = ['gold', 'green', 'coral', 'blue'], rot = 0, title ='Models performance (cross val mean)',edgecolor = 'grey', alpha = 0.5)

for p in ax.patches:

ax.annotate(str(p.get\_height()), (p.get\_x() \* 1.01, p.get\_height() \* 1.0005))

plt.show()

*Figure 1.2**6*

**1.10 Predictive Model 4: Deep Learning ANN Model using Tensorflow to predict**

Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons.A neuron takes a group of weighted inputs, applies an activation function, and returns an output.Inputs to a neuron can either be features from a training set or outputs from a previous layer’s neurons. Weights are applied to the inputs as they travel along synapses to reach the neuron. The neuron then applies an activation function to the “sum of weighted inputs” from each incoming synapse and passes the result on to all the neurons in the next layer.



*Figure 1.27*

**1.8.1 Data Pre-processing**

Data = pd.read\_csv("data.csv")

train,test = train\_test\_split(Data, test\_size=0.3, random\_state=42)

train\_id = train['id']

test\_id = test['id']

train\_data = train.iloc[:,1:]

test\_data = test.iloc[:,1:]

# Training Data

train\_x = train\_data.iloc[:,1:]

train\_x = MinMaxScaler().fit\_transform(train\_x)

# Testing Data

test\_x = test\_data.iloc[:,1:]

test\_x = MinMaxScaler().fit\_transform(test\_x)

# Training Data

train\_y = train\_data.iloc[:,:1]

train\_y[train\_y=='M'] = 0

train\_y[train\_y=='B'] = 1

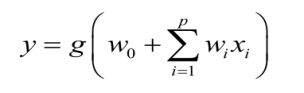
# Testing Data

test\_y = test\_data.iloc[:,:1]

test\_y[test\_y=='M'] = 0

test\_y[test\_y=='B'] = 1

****Output = Weight \* Input + Bias :**** For this equation, we already have output, input layers. But don't have weight value and bias value.Weight : a value that can give different weights depending on features and output , bias : a value that can give different weights depending on features



1>Placeholder for dynamic variable allocation Placeholder is one of the function in tensorflow.

for X, a place must have 30 columns, since wbcd data has 30 features.

for Y, a place must have 1 columns, since the results has 1 outcome.

If you see the row "None", it means it has no size limits.

X = tf.placeholder(tf.float32, [None,30])

Y = tf.placeholder(tf.float32, [None, 1])

2>Make Weight, Bias value with randomly

W(weight) : why **[30,1]**? 16 for 16 features, 1 for 1 Outcome(results).

P(weight): why **[10,1]**? 10 for 10 PCA features, 1 for 1 Outcome(results).

b(bias) : why **[1]**? outcome has 1 layers.

# weight

W = tf.Variable(tf.random\_normal([30,1], seed=0), name='weight')

# bias

b = tf.Variable(tf.random\_normal([1], seed=0), name='bias')

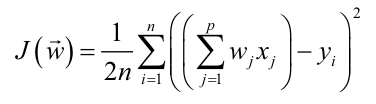
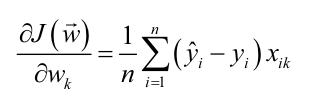
### 3>Outfut function of Artificial Neural Network[¶](http://localhost:8888/notebooks/Downloads/Book BPP/Breast Cancer/breast-cancer-analysis-and-prediction.ipynb" \l "Outfut-function-of-Artificial-Neural-Network)

* **Output = Weight \* Input + Bias**
* tf.matmul() : for array multiply

logits = tf.matmul(X,W) + b

4>Cross Entropy

For finding cost, you have to substract all blue dot value with red line. Next, You add all distance you find and get average. For good prediction, this average distance of red line & blue dot must be minimum value. tf.nn.sigmoid\_cross\_entropy\_with\_logits(): for gradient\_descent with sig results(hypothesis)



hypothesis = tf.nn.sigmoid(logits)

cost\_i = tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=logits,labels=Y)

cost = tf.reduce\_mean(cost\_i)

5>Gradient Descent Optimizer

* GradientDescentOptimizer: It makes the best result with the least error
* There are lots of optimizer methods provided in tensorflow. (GradientDescent, Adam, RMSProp, etc.)
* learning rate : It indicates the degree of descending size.

train = tf.train.GradientDescentOptimizer(learning\_rate=0.1).minimize(cost)

6>Difference between original vs. Prediction

prediction = tf.cast(hypothesis > 0.5, dtype=tf.float32)

correct\_prediction = tf.equal(prediction, Y)

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, dtype=tf.float32))

7>Activation Function

*Activation functions* are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable.*They* *introduce non-linear properties to our Network*.*Their main purpose is to convert a input signal of a node in a ANN to an output signal.* That output signal now is used as a input in the next layer in the stack.

**1.8.2 ANN & SLP**

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for step in range(10001):

sess.run(train, feed\_dict={X: train\_x, Y: train\_y})

if step % 1000 == 0:

loss, acc = sess.run([cost, accuracy], feed\_dict={X: train\_x, Y: train\_y})

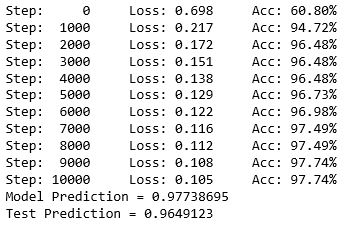
print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(step, loss, acc))

train\_acc = sess.run(accuracy, feed\_dict={X: train\_x, Y: train\_y})

test\_acc,test\_predict,test\_correct = sess.run([accuracy,prediction,correct\_prediction], feed\_dict={X: test\_x, Y: test\_y})

print("Model Prediction =", train\_acc)

print("Test Prediction =", test\_acc)

To compute anything, a graph must be launched in a session. Technically, session places the graph ops on hardware such as CPUs or GPUs and provides methods to execute them.

code creates a Session object (assigned to ****sess****), and then (the second line) invokes its run method to run enough of the computational graph to evaluate ****cost,accuracy.****here is no need to close the session at the end as it gets closed automatically.

*Figure 1.28*

**1.8.3 ANN - SLP -PCA Model**

pca\_train\_x, pca\_test\_x : normalization, PCA30 -> 10 features

train\_y, test\_y : we can use the same data as above activation values, since there are no changes in y\_data

def ann\_slp\_pca():

sklearn\_pca = sklearnPCA(n\_components=10)

print("===========Data Summary===========")

pca\_train\_x = sklearn\_pca.fit\_transform(train\_x)

print("PCA Training Data :", pca\_train\_x.shape)

pca\_test\_x = sklearn\_pca.fit\_transform(test\_x)

print("PCA Testing Data :", pca\_test\_x.shape)

X = tf.placeholder(tf.float32, [None,10])

Y = tf.placeholder(tf.float32, [None, 1])

W = tf.Variable(tf.random\_normal([10,1], seed=0), name='weight')

b = tf.Variable(tf.random\_normal([1], seed=0), name='bias')

logits = tf.matmul(X,W) + b

hypothesis = tf.nn.sigmoid(logits)

cost\_i = tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=logits,labels=Y)

cost = tf.reduce\_mean(cost\_i)

train = tf.train.GradientDescentOptimizer(learning\_rate=0.2).minimize(cost)

prediction = tf.cast(hypothesis > 0.5, dtype=tf.float32)

correct\_prediction = tf.equal(prediction, Y)

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, dtype=tf.float32))

print("\n============Processing============")

with tf.Session() as sess:

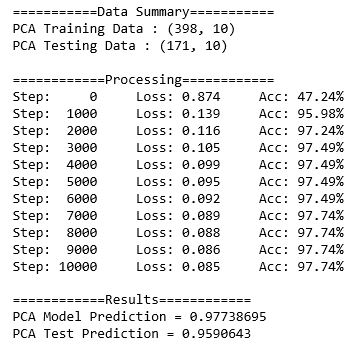
sess.run(tf.global\_variables\_initializer())

for step in range(10001):

sess.run(train, feed\_dict={X: pca\_train\_x, Y: train\_y})

if step % 1000 == 0:

loss, acc = sess.run([cost, accuracy], feed\_dict={X: pca\_train\_x, Y: train\_y})

 print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(step, loss, acc))

train\_acc = sess.run(accuracy, feed\_dict={X: pca\_train\_x, Y: train\_y})

test\_acc,test\_predict,test\_correct = sess.run([accuracy,prediction,correct\_prediction], feed\_dict={X: pca\_test\_x, Y: test\_y})

print("\n============Results============")

print("PCA Model Prediction =", train\_acc)

print("PCA Test Prediction =", test\_acc)

return train\_acc, test\_acc

ann\_slp\_pca\_train\_acc, ann\_slp\_pca\_test\_acc = ann\_slp\_pca()

*Figure 1.29*

**1.8.4 ANN - MLP Model**

Multi-Layer perceptron defines the most complicated architecture of artificial neural networks. It is substantially formed from multiple layers of perceptron.

def ann\_mlp():

print(“===========Data Summary===========")

print("Training Data :", train\_x.shape)

print("Testing Data :", test\_x.shape)

X = tf.placeholder(tf.float32, [None,30])

Y = tf.placeholder(tf.float32, [None, 1])

# input

W1 = tf.Variable(tf.random\_normal([30,60], seed=0), name='weight1')

b1 = tf.Variable(tf.random\_normal([60], seed=0), name='bias1')

layer1 = tf.nn.sigmoid(tf.matmul(X,W1) + b1)

# hidden1

W2 = tf.Variable(tf.random\_normal([60,60], seed=0), name='weight2')

b2 = tf.Variable(tf.random\_normal([60], seed=0), name='bias2')

layer2 = tf.nn.sigmoid(tf.matmul(layer1,W2) + b2)

# hidden2

W3 = tf.Variable(tf.random\_normal([60,90], seed=0), name='weight3')

b3 = tf.Variable(tf.random\_normal([90], seed=0), name='bias3')

layer3 = tf.nn.sigmoid(tf.matmul(layer2,W3) + b3)

# output

W4 = tf.Variable(tf.random\_normal([90,1], seed=0), name='weight4')

b4 = tf.Variable(tf.random\_normal([1], seed=0), name='bias4')

logits = tf.matmul(layer3,W4) + b4

hypothesis = tf.nn.sigmoid(logits)

cost\_i = tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=logits,labels=Y)

cost = tf.reduce\_mean(cost\_i)

train = tf.train.GradientDescentOptimizer(learning\_rate=0.001).minimize(cost)

prediction = tf.cast(hypothesis > 0.5, dtype=tf.float32)

correct\_prediction = tf.equal(prediction, Y)

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, dtype=tf.float32))

print("\n============Processing============")

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for step in range(10001):

sess.run(train, feed\_dict={X: train\_x, Y: train\_y})

if step % 1000 == 0:

loss, acc = sess.run([cost, accuracy], feed\_dict={X: train\_x, Y: train\_y})

print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(step, loss, acc))

train\_acc = sess.run(accuracy, feed\_dict={X: train\_x, Y: train\_y})

test\_acc,test\_predict,test\_correct = sess.run([accuracy,prediction,correct\_prediction], feed\_dict={X: test\_x, Y: test\_y})

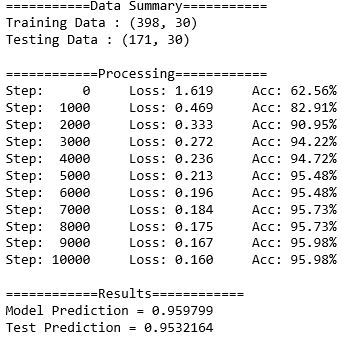
print("\n============Results============")

print("Model Prediction =", train\_acc)

print("Test Prediction =", test\_acc)

return train\_acc,test\_acc

ann\_mlp\_train\_acc, ann\_mlp\_test\_acc = ann\_mlp()

****

*Figure 1.29*

**1.9 Summary**

* Here we have learnt complete Exploratory Data Analysis with Various Statistical Analysis with Plotly.
* And we have got a familiarity various types plots with Plotly.
* Explored Various evaluation matrix for Machine learning and Deep learning.
* Here we have got to know how Logistic Regression,Random Forest, Ensemble Classifier works.
* And we have used various Deep learning Framework with ANN+PCA, ANN+PCA+MLP etc.
* We have learned how Tensorflow Frameworks works.