```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.tree import DecisionTreeClassifier as DTC
from matplotlib import pyplot as plt
%matplotlib inline
import datetime
import pandas as pd
from sklearn import preprocessing as pp
from sklearn.linear_model import LogisticRegression as LR
from sklearn.neighbors import KNeighborsClassifier as KNN

import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import np_utils

ds = pd.read_csv('mushrooms.csv')
dat = ds.values

print dat.shape

headers = list(ds.columns.values) #store features of mushrooms
```

```
(8124, 23)
```

```python
#IGNORE THIS CELL

arr = np.array([1,2,3])
b = np.array([4, 5, 6])

arr = np.vstack((arr, b))
#q = np.concatenate((arr,b), axis=0)
#print arr

le = pp.LabelEncoder()
le.fit(dat[:, 0])
y = le.transform(dat[:, 0])
le1 = pp.LabelEncoder()
le1.fit(dat[:, 1])
z = le1.transform(dat[:, 1])
le2 = pp.LabelEncoder()
le2.fit(dat[:, 2])
zz = le2.transform(dat[:, 2])

# y = np.vstack(y, z)
# z = np.vstack(z)
# y = np.vstack(y)
qqw = np.vstack((y, z))
qqw = np.vstack((qqw, zz))
#print qqw.T
```

```python
#Data Preprocessing
l = pp.LabelEncoder()
l.fit(dat[:, 0])
dataa = l.transform(dat[:, 0])

for ix in range(1, dat.shape[1]):
    le = pp.LabelEncoder()
    le.fit(dat[:, ix])
    y = le.transform(dat[:, ix])
    dataa = np.vstack((dataa , y))

data = dataa.T

cate = data[:, 0] #One hot encoding for Neural Network implementation

print data.shape ##FINALLLYY O YEAH!!!
```

```
(8124, 23)
```

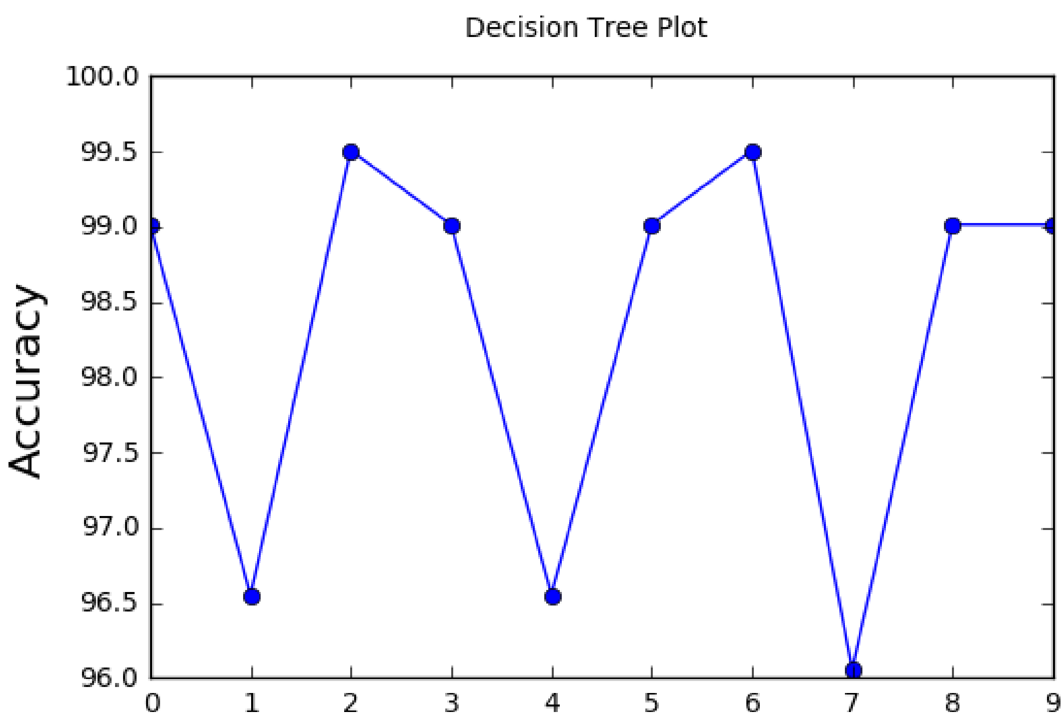
```python
split = int(0.80 * data.shape[0])

x_train = data[:split , 1:]
y_train = data[:split, 0]

x_test = data[split: , 1:]
y_test = data[split: , 0]

print x_train.shape, y_train.shape
print x_test.shape, y_test.shape
```

```
(6499, 22) (6499,)
(1625, 22) (1625,)
```

```python
acc = []
ans = []

for ix in range(10):
    dt = DTC()

    start = datetime.datetime.now()
    dt.fit(x_train, y_train)
    end = datetime.datetime.now()

    #print "Training Time : ", end-start

    start = datetime.datetime.now()
    score = dt.score(x_test, y_test)
    end = datetime.datetime.now()

    #print "Testing Time : ", end-start
```

```
    #print "Accurcy : ", score*100
    acc.append(score*100)
    tem = dt.feature_importances_
    ans.append(tem)
    #print "\n"

temp = []
for ix in range(0, len(ans)):
    temp.append(np.argmax(ans[ix]))

mode = max(set(temp), key=temp.count) #find mode for features
importance in decision trees
print "Features most indicative of a poisonous mushroom wrt Decision
Tree Model : ", headers[mode+1]

plt.figure(0)
plt.suptitle('Decision Tree Plot', fontsize=10)
plt.plot(acc, '-o')
plt.ylabel('Accuracy', fontsize=16)
plt.show()
```

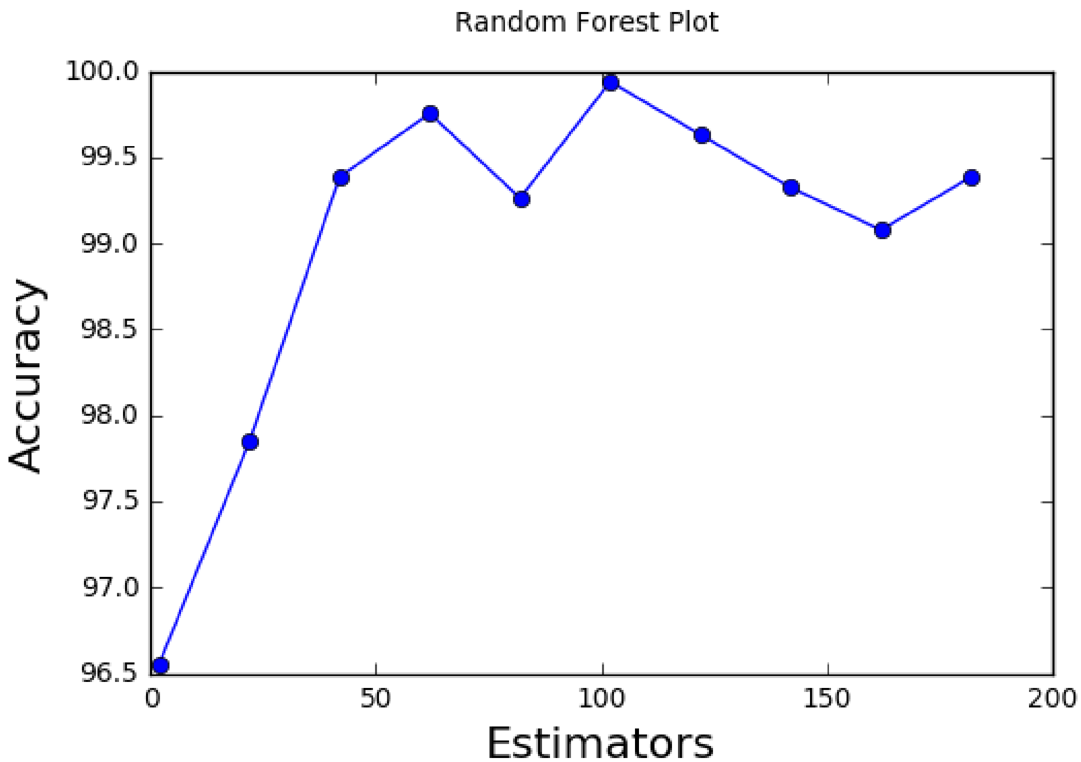
```
Features most indicative of a poisonous mushroom wrt Decision Tree
Model :  spore-print-color
```




```
est = [] #taking variable estimators
acc = []
ans = []
```

```
for iy in range(2, 200, 20):
    est.append(iy)

for ix in range(len(est)):
    rf = RFC(n_estimators=est[ix], n_jobs=2)
    #print "No. of Decision Trees : ", est[ix]

    start = datetime.datetime.now()
    rf.fit(x_train, y_train)
    end = datetime.datetime.now()

    #print "Training Time : ", end-start

    start = datetime.datetime.now()
    score = rf.score(x_test, y_test)
    end = datetime.datetime.now()

    #print "Testing Time : ", end-start

    #print "Accurcy : ", score*100
    acc.append(score*100)
    tem = rf.feature_importances_
    ans.append(tem)
    #print "\n"

temp = []

for ix in range(0, len(ans)):
    temp.append(np.argmax(ans[ix]))

mode = max(set(temp), key=temp.count) #find mode for features
importance in variable estimators
print "Features most indicative of a poisonous mushroom wrt Random
Forest Model : ", headers[mode+1]

plt.figure(1)
plt.suptitle('Random Forest Plot', fontsize=10)
plt.plot(est, acc, '-o')
plt.xlabel('Estimators', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.show()
```

```
Features most indicative of a poisonous mushroom wrt Random Forest
Model :  odor
```




```
lr = LR(n_jobs=-1)

start = datetime.datetime.now()
lr.fit(x_train, y_train)
end = datetime.datetime.now()

print "Training Time : ", end-start

start = datetime.datetime.now()
score = lr.score(x_test, y_test)
end = datetime.datetime.now()

print "Testing Time : ", end-start

print "Accurcy : ", score*100

features = dt.feature_importances_
#print "\n"
```

```
Training Time :  0:00:00.108130
Testing Time :  0:00:00.000907
Accurcy :  89.4153846154
```

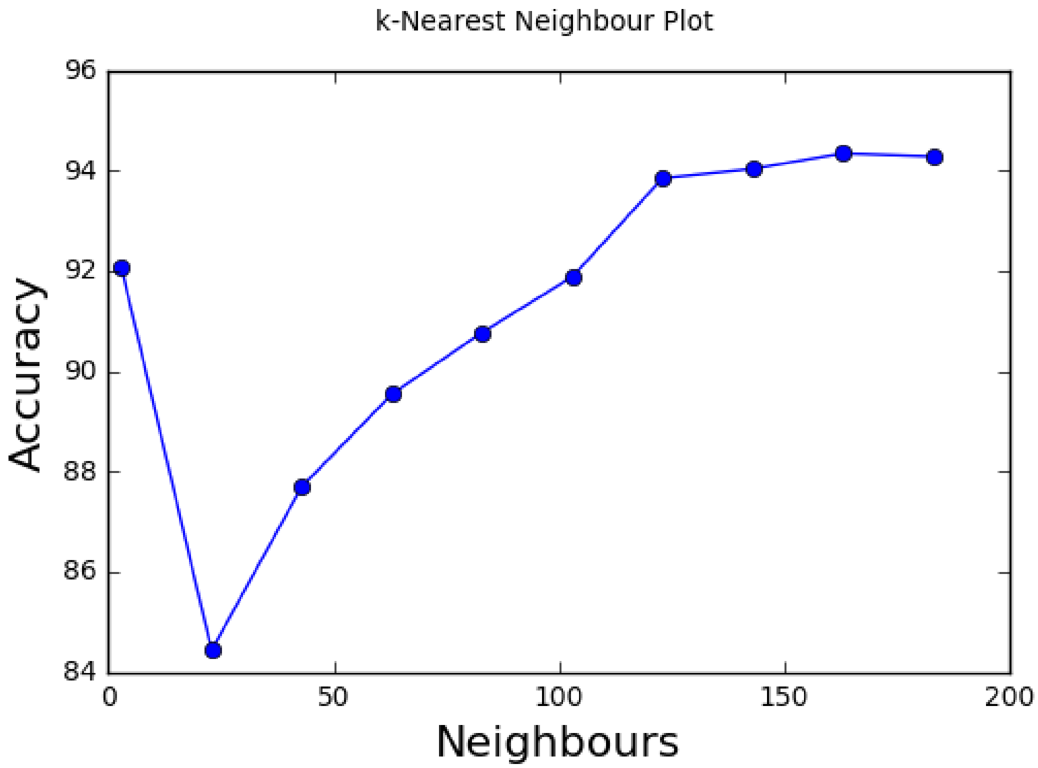
```
temp = np.argmax(features)
```

```
print "Features most indicative of a poisonous mushroom wrt Logistic
Regression Model : ", headers[temp+1]
```

```
Features most indicative of a poisonous mushroom wrt Logistic
Regression Model :  spore-print-color
```

```
acc = []
ans = []
neighbours = []

for ix in range(3, 200, 20):
    neighbours.append(ix)

for ix in range(len(neighbours)):
    knn = KNN(n_neighbors=neighbours[ix], n_jobs=-1)

    start = datetime.datetime.now()
    knn.fit(x_train, y_train)
    end = datetime.datetime.now()

    #print "Training Time : ", end-start

    start = datetime.datetime.now()
    score = knn.score(x_test, y_test)
    end = datetime.datetime.now()

    #print "Testing Time : ", end-start

    #print "Accurcy : ", score*100
    acc.append(score*100)
    temp = dt.feature_importances_
    ans.append(temp)
    #print "\n"

temp = []

for ix in range(0, len(ans)):
    temp.append(np.argmax(ans[ix]))

mode = max(set(temp), key=temp.count) #find mode for features
importance in variable estimators
print "Features most indicative of a poisonous mushroom wrt kNN : ",
headers[mode+1]

plt.figure(2)
plt.suptitle('k-Nearest Neighbour Plot', fontsize=10)
plt.plot(neighbours, acc, '-o')
plt.xlabel('Neighbours', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.show()
```

```
Features most indicative of a poisonous mushroom wrt kNN :  spore-
print-color
```




```
y = np_utils.to_categorical(cate)
Y_train = y[:split]
Y_test = y[split:]

print x_train.shape, x_test.shape
print y_train.shape, y_test.shape
print data.shape
```

```
(6499, 22) (1625, 22)
(6499,) (1625,)
(8124, 23)
```

```
model = Sequential()

model.add(Dense(11, input_shape=(22,)))
model.add(Activation('relu'))

model.add(Dense(5))
model.add(Activation('relu'))

model.add(Dense(2))
model.add(Activation('softmax'))
```

```
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
________________________________________________________________________________
______________________________
Layer (type)                     Output Shape          Param #
Connected to
================================================================================
==============================
dense_46 (Dense)                 (None, 11)            253
dense_input_16[0][0]
________________________________________________________________________________
______________________________
activation_46 (Activation)       (None, 11)            0
dense_46[0][0]
________________________________________________________________________________
______________________________
dense_47 (Dense)                 (None, 5)             60
activation_46[0][0]
________________________________________________________________________________
______________________________
activation_47 (Activation)       (None, 5)             0
dense_47[0][0]
________________________________________________________________________________
______________________________
dense_48 (Dense)                 (None, 2)             12
activation_47[0][0]
________________________________________________________________________________
______________________________
activation_48 (Activation)       (None, 2)             0
dense_48[0][0]
================================================================================
==============================
Total params: 325
________________________________________________________________________________
______________________________
```

```
hist = model.fit(x_train, Y_train,
         nb_epoch=70,
         shuffle=True,
         batch_size=128,
         validation_data=(x_test, Y_test))
```

```
Train on 6499 samples, validate on 1625 samples
Epoch 1/70
6499/6499 [==============================] - 0s - loss: 0.7276 - acc:
0.5890 - val_loss: 0.7540 - val_acc: 0.4308
Epoch 2/70
6499/6499 [==============================] - 0s - loss: 0.5810 - acc:
0.6916 - val_loss: 0.6043 - val_acc: 0.6880
```

```
Epoch 3/70
6499/6499 [==============================] - 0s - loss: 0.5132 - acc:
0.7617 - val_loss: 0.4923 - val_acc: 0.8338
Epoch 4/70
6499/6499 [==============================] - 0s - loss: 0.4594 - acc:
0.8070 - val_loss: 0.4260 - val_acc: 0.8406
Epoch 5/70
6499/6499 [==============================] - 0s - loss: 0.4140 - acc:
0.8297 - val_loss: 0.3927 - val_acc: 0.8425
Epoch 6/70
6499/6499 [==============================] - 0s - loss: 0.3762 - acc:
0.8487 - val_loss: 0.3678 - val_acc: 0.8769
Epoch 7/70
6499/6499 [==============================] - 0s - loss: 0.3423 - acc:
0.8667 - val_loss: 0.3530 - val_acc: 0.8929
Epoch 8/70
6499/6499 [==============================] - 0s - loss: 0.3138 - acc:
0.8808 - val_loss: 0.3414 - val_acc: 0.8911
Epoch 9/70
6499/6499 [==============================] - 0s - loss: 0.2841 - acc:
0.8929 - val_loss: 0.3373 - val_acc: 0.8738
Epoch 10/70
6499/6499 [==============================] - 0s - loss: 0.2520 - acc:
0.9097 - val_loss: 0.3347 - val_acc: 0.8751
Epoch 11/70
6499/6499 [==============================] - 0s - loss: 0.2169 - acc:
0.9257 - val_loss: 0.3476 - val_acc: 0.8578
Epoch 12/70
6499/6499 [==============================] - 0s - loss: 0.1863 - acc:
0.9368 - val_loss: 0.3444 - val_acc: 0.8578
Epoch 13/70
6499/6499 [==============================] - 0s - loss: 0.1633 - acc:
0.9454 - val_loss: 0.3321 - val_acc: 0.8726
Epoch 14/70
6499/6499 [==============================] - 0s - loss: 0.1445 - acc:
0.9528 - val_loss: 0.3393 - val_acc: 0.8683
Epoch 15/70
6499/6499 [==============================] - 0s - loss: 0.1300 - acc:
0.9594 - val_loss: 0.3327 - val_acc: 0.8751
Epoch 16/70
6499/6499 [==============================] - 0s - loss: 0.1175 - acc:
0.9625 - val_loss: 0.3382 - val_acc: 0.8702
Epoch 17/70
6499/6499 [==============================] - 0s - loss: 0.1076 - acc:
0.9655 - val_loss: 0.3317 - val_acc: 0.8751
Epoch 18/70
6499/6499 [==============================] - 0s - loss: 0.0999 - acc:
0.9666 - val_loss: 0.3184 - val_acc: 0.8855
Epoch 19/70
6499/6499 [==============================] - 0s - loss: 0.0929 - acc:
```

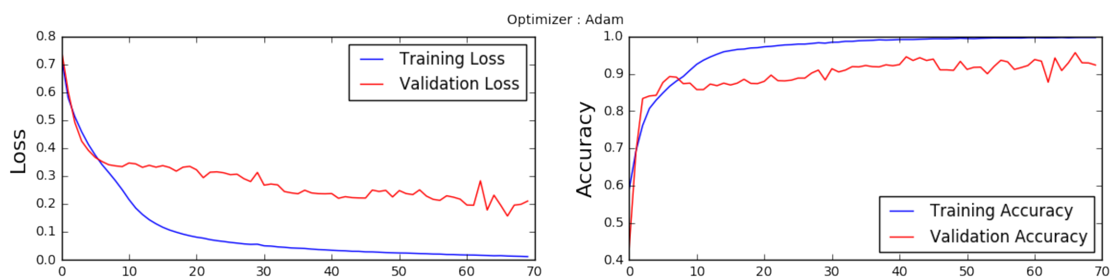
```
0.9695 - val_loss: 0.3326 - val_acc: 0.8745
Epoch 20/70
6499/6499 [==============================] - 0s - loss: 0.0870 - acc:
0.9703 - val_loss: 0.3356 - val_acc: 0.8738
Epoch 21/70
6499/6499 [==============================] - 0s - loss: 0.0820 - acc:
0.9728 - val_loss: 0.3232 - val_acc: 0.8800
Epoch 22/70
6499/6499 [==============================] - 0s - loss: 0.0785 - acc:
0.9738 - val_loss: 0.2950 - val_acc: 0.8966
Epoch 23/70
6499/6499 [==============================] - 0s - loss: 0.0733 - acc:
0.9758 - val_loss: 0.3146 - val_acc: 0.8818
Epoch 24/70
6499/6499 [==============================] - 0s - loss: 0.0695 - acc:
0.9777 - val_loss: 0.3160 - val_acc: 0.8812
Epoch 25/70
6499/6499 [==============================] - 0s - loss: 0.0667 - acc:
0.9783 - val_loss: 0.3126 - val_acc: 0.8831
Epoch 26/70
6499/6499 [==============================] - 0s - loss: 0.0633 - acc:
0.9800 - val_loss: 0.3058 - val_acc: 0.8886
Epoch 27/70
6499/6499 [==============================] - 0s - loss: 0.0608 - acc:
0.9800 - val_loss: 0.3074 - val_acc: 0.8886
Epoch 28/70
6499/6499 [==============================] - 0s - loss: 0.0580 - acc:
0.9815 - val_loss: 0.2915 - val_acc: 0.9022
Epoch 29/70
6499/6499 [==============================] - 0s - loss: 0.0563 - acc:
0.9835 - val_loss: 0.2811 - val_acc: 0.9102
Epoch 30/70
6499/6499 [==============================] - 0s - loss: 0.0569 - acc:
0.9825 - val_loss: 0.3139 - val_acc: 0.8837
Epoch 31/70
6499/6499 [==============================] - 0s - loss: 0.0511 - acc:
0.9849 - val_loss: 0.2683 - val_acc: 0.9138
Epoch 32/70
6499/6499 [==============================] - 0s - loss: 0.0501 - acc:
0.9852 - val_loss: 0.2723 - val_acc: 0.9052
Epoch 33/70
6499/6499 [==============================] - 0s - loss: 0.0474 - acc:
0.9874 - val_loss: 0.2693 - val_acc: 0.9102
Epoch 34/70
6499/6499 [==============================] - 0s - loss: 0.0458 - acc:
0.9874 - val_loss: 0.2454 - val_acc: 0.9194
Epoch 35/70
6499/6499 [==============================] - 0s - loss: 0.0436 - acc:
0.9889 - val_loss: 0.2407 - val_acc: 0.9188
Epoch 36/70
```

```
6499/6499 [==============================] - 0s - loss: 0.0425 - acc:
0.9895 - val_loss: 0.2374 - val_acc: 0.9225
Epoch 37/70
6499/6499 [==============================] - 0s - loss: 0.0416 - acc:
0.9902 - val_loss: 0.2502 - val_acc: 0.9194
Epoch 38/70
6499/6499 [==============================] - 0s - loss: 0.0394 - acc:
0.9914 - val_loss: 0.2406 - val_acc: 0.9188
Epoch 39/70
6499/6499 [==============================] - 0s - loss: 0.0377 - acc:
0.9908 - val_loss: 0.2378 - val_acc: 0.9243
Epoch 40/70
6499/6499 [==============================] - 0s - loss: 0.0368 - acc:
0.9912 - val_loss: 0.2374 - val_acc: 0.9218
Epoch 41/70
6499/6499 [==============================] - 0s - loss: 0.0351 - acc:
0.9922 - val_loss: 0.2381 - val_acc: 0.9249
Epoch 42/70
6499/6499 [==============================] - 0s - loss: 0.0339 - acc:
0.9923 - val_loss: 0.2213 - val_acc: 0.9458
Epoch 43/70
6499/6499 [==============================] - 0s - loss: 0.0333 - acc:
0.9923 - val_loss: 0.2271 - val_acc: 0.9360
Epoch 44/70
6499/6499 [==============================] - 0s - loss: 0.0314 - acc:
0.9931 - val_loss: 0.2237 - val_acc: 0.9434
Epoch 45/70
6499/6499 [==============================] - 0s - loss: 0.0312 - acc:
0.9935 - val_loss: 0.2226 - val_acc: 0.9360
Epoch 46/70
6499/6499 [==============================] - 0s - loss: 0.0296 - acc:
0.9943 - val_loss: 0.2222 - val_acc: 0.9397
Epoch 47/70
6499/6499 [==============================] - 0s - loss: 0.0293 - acc:
0.9943 - val_loss: 0.2507 - val_acc: 0.9108
Epoch 48/70
6499/6499 [==============================] - 0s - loss: 0.0281 - acc:
0.9942 - val_loss: 0.2457 - val_acc: 0.9108
Epoch 49/70
6499/6499 [==============================] - 0s - loss: 0.0269 - acc:
0.9948 - val_loss: 0.2497 - val_acc: 0.9095
Epoch 50/70
6499/6499 [==============================] - 0s - loss: 0.0262 - acc:
0.9957 - val_loss: 0.2261 - val_acc: 0.9335
Epoch 51/70
6499/6499 [==============================] - 0s - loss: 0.0254 - acc:
0.9951 - val_loss: 0.2489 - val_acc: 0.9120
Epoch 52/70
6499/6499 [==============================] - 0s - loss: 0.0251 - acc:
0.9946 - val_loss: 0.2381 - val_acc: 0.9175
```

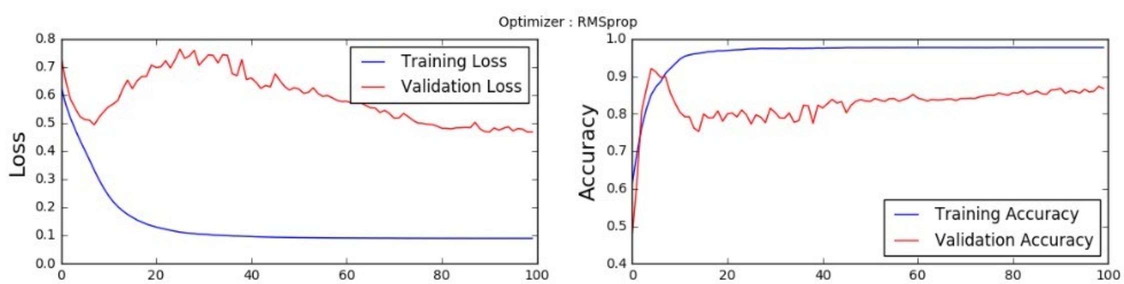
```
Epoch 53/70
6499/6499 [==============================] - 0s - loss: 0.0242 - acc:
0.9955 - val_loss: 0.2345 - val_acc: 0.9182
Epoch 54/70
6499/6499 [==============================] - 0s - loss: 0.0233 - acc:
0.9960 - val_loss: 0.2515 - val_acc: 0.9003
Epoch 55/70
6499/6499 [==============================] - 0s - loss: 0.0228 - acc:
0.9965 - val_loss: 0.2293 - val_acc: 0.9200
Epoch 56/70
6499/6499 [==============================] - 0s - loss: 0.0216 - acc:
0.9963 - val_loss: 0.2175 - val_acc: 0.9366
Epoch 57/70
6499/6499 [==============================] - 0s - loss: 0.0213 - acc:
0.9963 - val_loss: 0.2137 - val_acc: 0.9323
Epoch 58/70
6499/6499 [==============================] - 0s - loss: 0.0200 - acc:
0.9965 - val_loss: 0.2302 - val_acc: 0.9120
Epoch 59/70
6499/6499 [==============================] - 0s - loss: 0.0198 - acc:
0.9963 - val_loss: 0.2253 - val_acc: 0.9169
Epoch 60/70
6499/6499 [==============================] - 0s - loss: 0.0188 - acc:
0.9972 - val_loss: 0.2182 - val_acc: 0.9225
Epoch 61/70
6499/6499 [==============================] - 0s - loss: 0.0182 - acc:
0.9974 - val_loss: 0.1971 - val_acc: 0.9385
Epoch 62/70
6499/6499 [==============================] - 0s - loss: 0.0179 - acc:
0.9969 - val_loss: 0.1963 - val_acc: 0.9342
Epoch 63/70
6499/6499 [==============================] - 0s - loss: 0.0171 - acc:
0.9968 - val_loss: 0.2839 - val_acc: 0.8775
Epoch 64/70
6499/6499 [==============================] - 0s - loss: 0.0165 - acc:
0.9971 - val_loss: 0.1792 - val_acc: 0.9428
Epoch 65/70
6499/6499 [==============================] - 0s - loss: 0.0155 - acc:
0.9977 - val_loss: 0.2325 - val_acc: 0.9089
Epoch 66/70
6499/6499 [==============================] - 0s - loss: 0.0159 - acc:
0.9969 - val_loss: 0.1963 - val_acc: 0.9298
Epoch 67/70
6499/6499 [==============================] - 0s - loss: 0.0145 - acc:
0.9975 - val_loss: 0.1578 - val_acc: 0.9569
Epoch 68/70
6499/6499 [==============================] - 0s - loss: 0.0139 - acc:
0.9977 - val_loss: 0.1967 - val_acc: 0.9298
Epoch 69/70
6499/6499 [==============================] - 0s - loss: 0.0133 - acc:
```

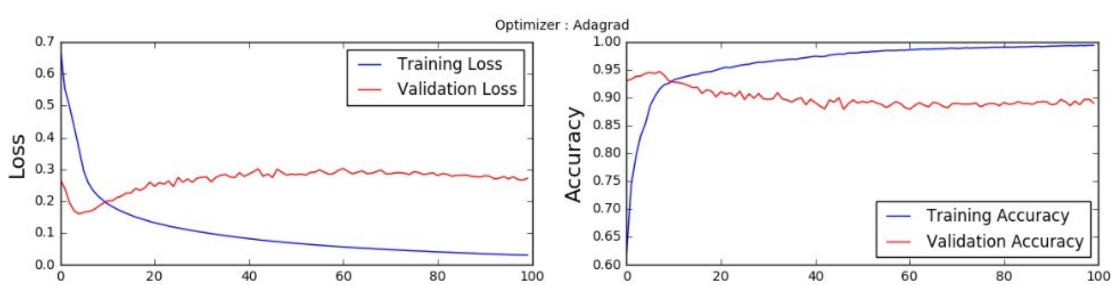
```
0.9977 - val_loss: 0.1997 - val_acc: 0.9292
Epoch 70/70
6499/6499 [==============================] - 0s - loss: 0.0124 - acc:
0.9977 - val_loss: 0.2114 - val_acc: 0.9237
```

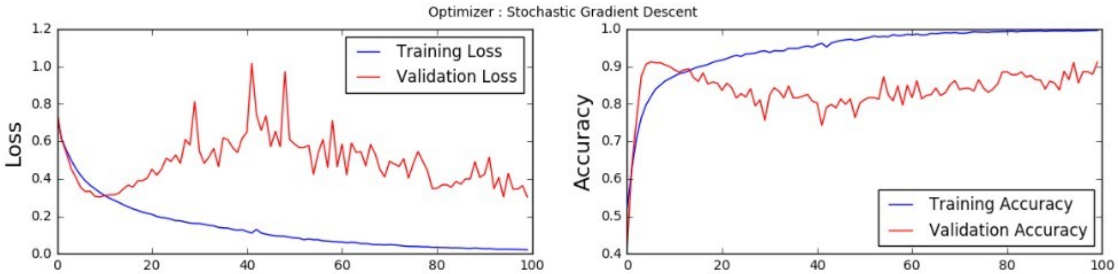
```
plt.figure(figsize=(14,3))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(hist.history['loss'], 'b', label='Training Loss')
plt.plot(hist.history['val_loss'], 'r', label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(hist.history['acc'], 'b', label='Training Accuracy')
plt.plot(hist.history['val_acc'], 'r', label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()
```



Plots with different Optimizers are ploted below







Optimizer : Stochastic Gradient Descent
Loss
Training Loss
Validation Loss
Accuracy
Training Accuracy
Validation Accuracy