Classify different data sets

Basic includes

In [2]:

```
# Using pandas to load the csv file
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt

from keras import models
from keras import layers
from keras import callbacks
from keras.utils import to_categorical

# reuters and fashin mnist data set from keras
from keras.datasets import reuters
from keras.datasets import fashion_mnist

# needed to preprocess text
from keras.preprocessing.text import Tokenizer
```

executed in 3.88s, finished 13:44:52 2019-02-19

```
Using TensorFlow backend.
```

Classify the Fashion Mnist

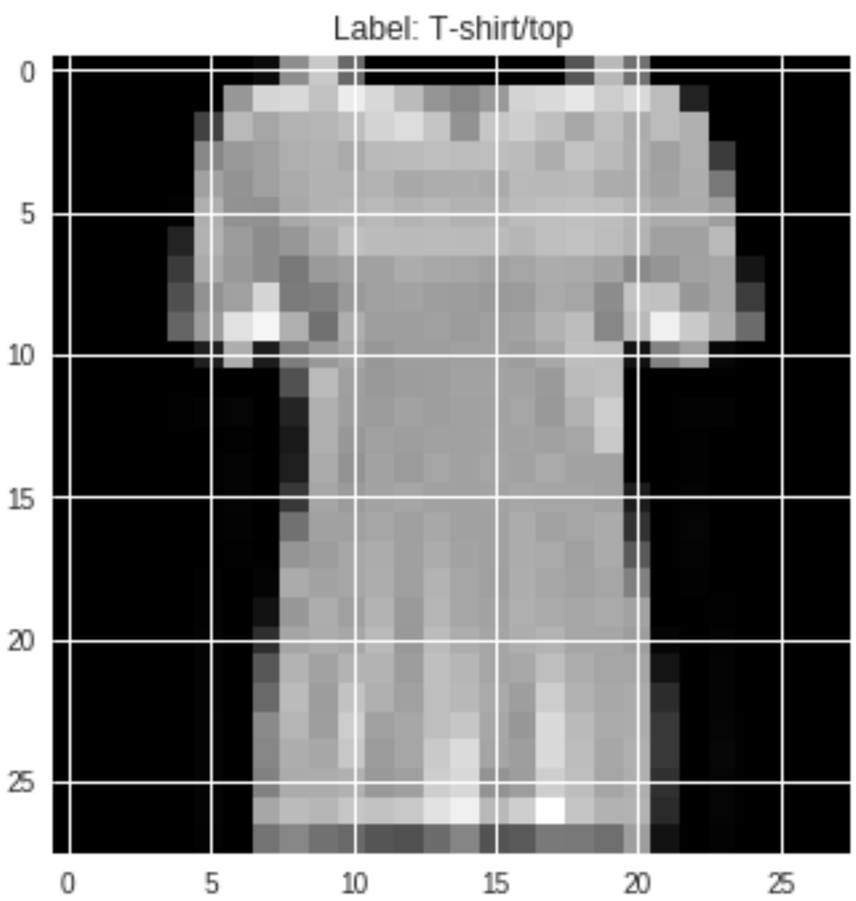
In [86]:

```
(fashion_train_data, fashion_train_labels), (fashion_test_data, fashion_test_l
fashion_class_labels = [
    'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt',
]
print(fashion_train_data.shape)

test_index = 10

plt.title("Label: " + fashion_class_labels[fashion_train_labels[test_index]])
plt.imshow(fashion_train_data[test_index], cmap="gray")
```

```
(60000, 28, 28)
```

Out[86]:

```
<matplotlib.image.AxesImage at 0x7facef2f0940>
```



TO DO: Preprocess the data

1. Normalize the input data set
2. Perform one hot encoding
3. Create a train, test, and validation set

In [0]:

```
# Normalize the input data set
# flatten images
fashion_train_data = fashion_train_data.reshape((60000, 784))
fashion_train_data = fashion_train_data.astype('float32') / 255

fashion_test_data = fashion_test_data.reshape((10000, 784))
fashion_test_data = fashion_test_data.astype('float32') / 255

# one hot encoding
fashion_train_labels = to_categorical(fashion_train_labels)
fashion_test_labels = to_categorical(fashion_test_labels)

validation_set_labels = fashion_train_labels[50000:]
validation_set = fashion_train_data[50000:]

training_set_labels = fashion_train_labels[:50000]
training_set = fashion_train_data[:50000]
```

TO DO: Define and train a network, then plot the accuracy of the training, validation, and testing

1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

In [88]:

```python
# Crate NN

nn_fashion_model = models.Sequential()
fashion_dropout = 0.3

nn_fashion_model.add(layers.Dense(1024, activation= "relu", input_shape= (784,

nn_fashion_model.add(layers.Dropout(fashion_dropout))

nn_fashion_model.add(layers.Dense(256, activation="relu"))
nn_fashion_model.add(layers.Dense(128, activation="relu"))

nn_fashion_model.add(layers.Dropout(fashion_dropout))

# Last layer, same size has the number of categories
nn_fashion_model.add(layers.Dense(10, activation="softmax"))



nn_fashion_early_stops = [
    callbacks.EarlyStopping(monitor= 'val_loss', patience= 4)
]

nn_fashion_model.compile(
    loss= "categorical_crossentropy", optimizer= "adam", metrics= ["accuracy"]
)

nn_fashion_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_47 (Dense)             (None, 1024)              803840    
_________________________________________________________________
dropout_17 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_48 (Dense)             (None, 256)               262400    
_________________________________________________________________
dense_49 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_50 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,100,426
Trainable params: 1,100,426
Non-trainable params: 0
_________________________________________________________________
```

In [89]:

```
# Train the NN model
fashion_epochs = 16
nn_fashion_history = nn_fashion_model.fit(
    fashion_train_data,
    fashion_train_labels,
    batch_size= 1024,
    epochs= fashion_epochs,
    verbose= 2,
    callbacks= nn_fashion_early_stops,
    validation_data= (validation_set, validation_set_labels)
)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/16
 - 8s - loss: 0.7594 - acc: 0.7338 - val_loss: 0.4379 - val_acc: 0.843
4
Epoch 2/16
 - 7s - loss: 0.4523 - acc: 0.8404 - val_loss: 0.3780 - val_acc: 0.862
7
Epoch 3/16
 - 7s - loss: 0.3956 - acc: 0.8588 - val_loss: 0.3368 - val_acc: 0.877
7
Epoch 4/16
 - 7s - loss: 0.3634 - acc: 0.8692 - val_loss: 0.3245 - val_acc: 0.881
2
Epoch 5/16
 - 7s - loss: 0.3418 - acc: 0.8758 - val_loss: 0.3016 - val_acc: 0.889
5
Epoch 6/16
 - 8s - loss: 0.3206 - acc: 0.8832 - val_loss: 0.2827 - val_acc: 0.895
7
Epoch 7/16
 - 8s - loss: 0.3106 - acc: 0.8865 - val_loss: 0.2850 - val_acc: 0.894
9
Epoch 8/16
 - 7s - loss: 0.3032 - acc: 0.8897 - val_loss: 0.2718 - val_acc: 0.894
9
Epoch 9/16
 - 7s - loss: 0.2951 - acc: 0.8915 - val_loss: 0.2719 - val_acc: 0.897
2
Epoch 10/16
 - 8s - loss: 0.2813 - acc: 0.8959 - val_loss: 0.2448 - val_acc: 0.910
5
Epoch 11/16
 - 7s - loss: 0.2728 - acc: 0.8990 - val_loss: 0.2459 - val_acc: 0.905
0
Epoch 12/16
 - 7s - loss: 0.2696 - acc: 0.8997 - val_loss: 0.2335 - val_acc: 0.910
1
```

```
Epoch 13/16
 - 7s - loss: 0.2591 - acc: 0.9043 - val_loss: 0.2241 - val_acc: 0.914
8
Epoch 14/16
 - 7s - loss: 0.2564 - acc: 0.9056 - val_loss: 0.2170 - val_acc: 0.917
0
Epoch 15/16
 - 7s - loss: 0.2504 - acc: 0.9066 - val_loss: 0.2166 - val_acc: 0.917
3
Epoch 16/16
 - 7s - loss: 0.2433 - acc: 0.9080 - val_loss: 0.2176 - val_acc: 0.917
3
```

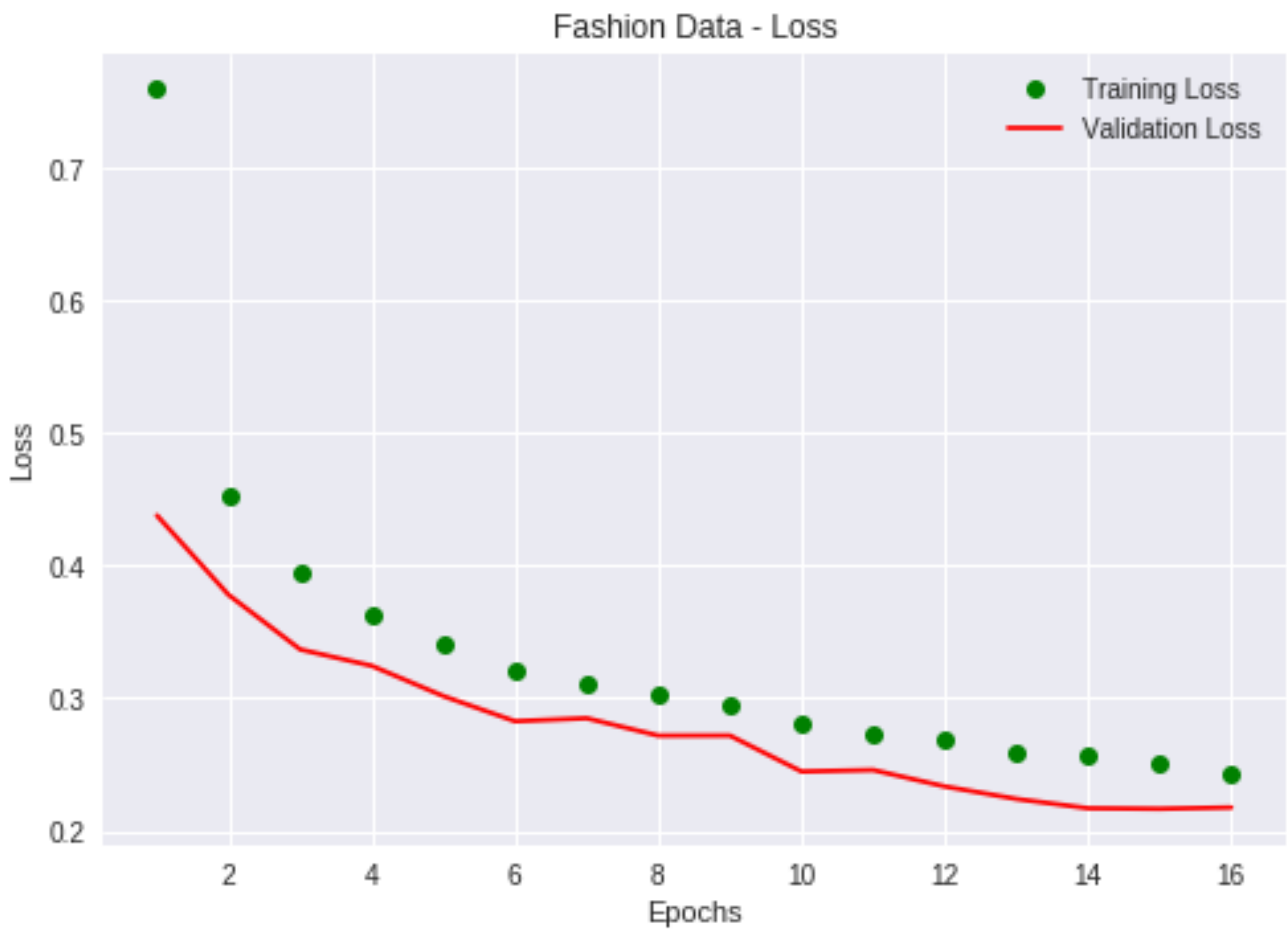
In [90]:

```python
fashion_result = nn_fashion_model.evaluate(fashion_test_data, fashion_test_lab
print('Fashion score: {}%'.format(fashion_result[1]*100))
```

```
10000/10000 [==============================] - 1s 105us/step
Fashion score: 88.74%
```

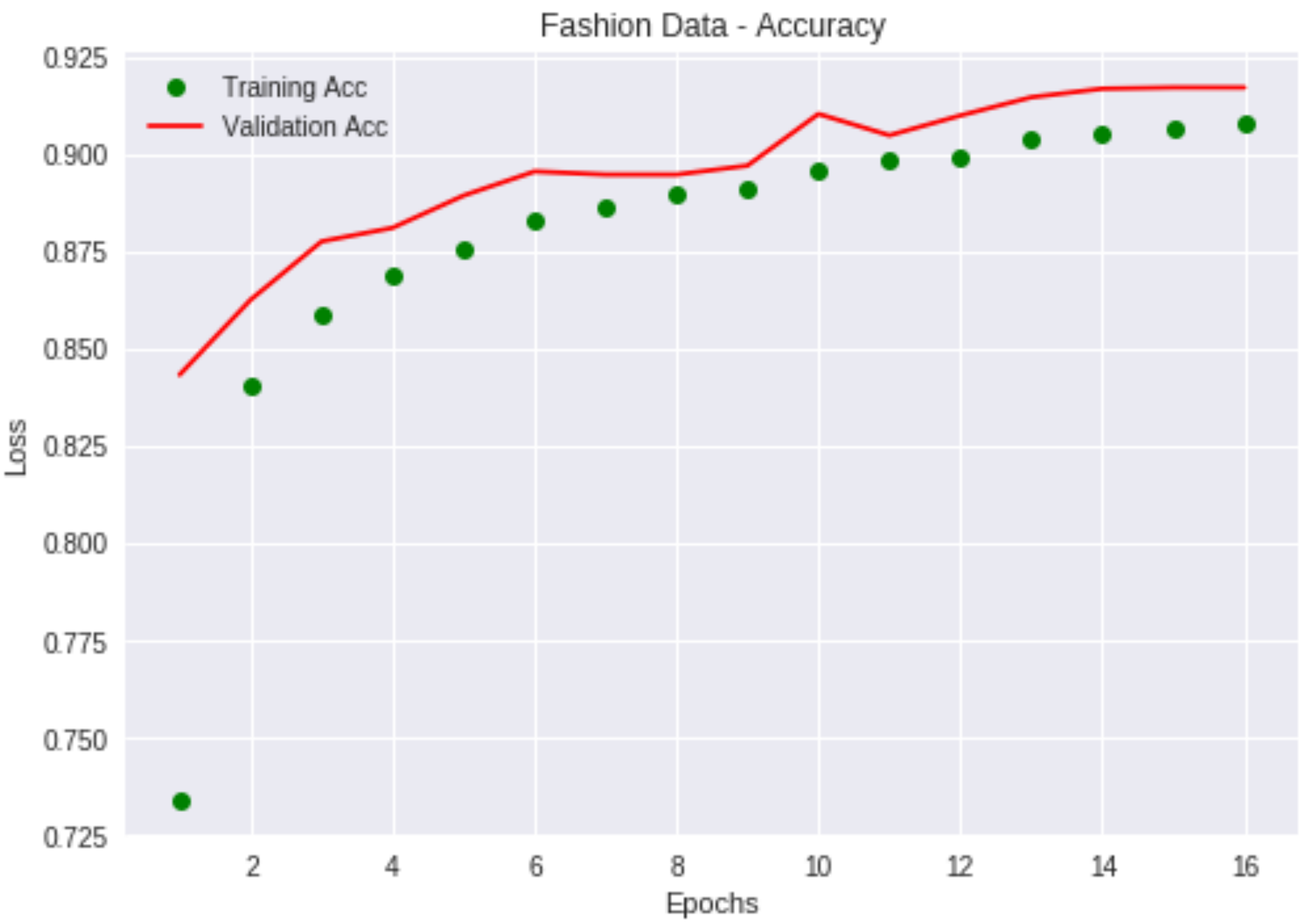
In [91]:

```
fashion_history  = nn_fashion_history.history
fashion_loss     = fashion_history['loss']
fashion_val_loss = fashion_history['val_loss']
fashion_epochs   = range(1, len(fashion_loss) + 1)

plt.plot(fashion_epochs, fashion_loss, 'go', label='Training Loss')
plt.plot(fashion_epochs, fashion_val_loss, 'r', label='Validation Loss')

plt.title('Fashion Data - Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [92]:

```
fashion_acc     = fashion_history['acc']
fashion_val_acc = fashion_history['val_acc']

plt.plot(fashion_epochs, fashion_acc, 'go', label='Training Acc')
plt.plot(fashion_epochs, fashion_val_acc, 'r', label='Validation Acc')

plt.title('Fashion Data - Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



Fashion conclusion

> In this model I first did over fitting so making the model network complex, from that point I moved some hyper parameters and successfully achieved a score above 85%.

Classifying newswires

Build a network to classify Reuters newswires into 46 different mutually-exclusive topics.

Load and review the data

In [93]:

```
reuters_max_words = 10000
(reuters_train_data, reuters_train_labels), (reuters_test_data, reuters_test_l

print(reuters_train_data.shape)
print(reuters_train_labels.shape)
print(reuters_train_data[0])
print(reuters_train_labels[0])

print(set(reuters_train_labels))
```

```
(8982,)
(8982,)
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186,
90, 67, 7, 89, 5, 19, 102, 6, 19, 124, 15, 90, 67, 84, 22, 482, 26, 7,
48, 4, 49, 8, 864, 39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 1
5, 7, 48, 9, 4579, 1005, 504, 6, 258, 6, 272, 11, 15, 22, 134, 44, 11,
15, 16, 8, 197, 1245, 90, 67, 52, 29, 209, 30, 32, 132, 6, 109, 15, 17
, 12]
3
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37
, 38, 39, 40, 41, 42, 43, 44, 45}
```

Load the word index to decode the train data.

In [94]:

```
word_index = reuters.get_word_index()

reverse_index = dict([(value+3, key) for (key, value) in word_index.items()])

reverse_index[0] = "<PAD>"
reverse_index[1] = "<START>"
reverse_index[2] = "<UNKNOWN>"  # unknown
reverse_index[3] = "<UNUSED>"

decoded_review = ' '.join([reverse_index.get(i,'?') for i in reuters_train_dat

print(decoded_review)
```

```
<START> <UNKNOWN> <UNKNOWN> said as a result of its december acquisiti
on of space co it expects earnings per share in 1987 of 1 15 to 1 30 d
lrs per share up from 70 cts in 1986 the company said pretax net shoul
d rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental ope
ration revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash f
low per share this year should be 2 50 to three dlrs reuter 3
```

TO DO: Preprocess the data

1. Normalize the input data set
2. Perform one hot encoding
3. Create a train, test, and validation set

In [95]:

```
tokenizer = Tokenizer(num_words= reuters_max_words)

reuters_train_data_token = tokenizer.sequences_to_matrix(
    reuters_train_data, mode="binary"
)
reuters_test_data_token = tokenizer.sequences_to_matrix(
    reuters_test_data, mode="binary"
)

reuters_one_hot_train_labels = to_categorical(reuters_train_labels)
reuters_one_hot_test_labels  = to_categorical(reuters_test_labels)

reuters_val_data   = reuters_train_data_token[:1000]
reuters_val_labels = reuters_one_hot_train_labels[:1000]

reuters_train_data   = reuters_train_data_token[1000:]
reuters_train_labels = reuters_one_hot_train_labels[1000:]

print('train:')
print(reuters_train_data.shape)
print(reuters_train_labels.shape)
print('val:')
print(reuters_val_data.shape)
print(reuters_val_labels.shape)
print('test:')
print(reuters_test_data_token.shape)
print(reuters_one_hot_test_labels.shape)
```

```
train:
(7982, 10000)
(7982, 46)
val:
(1000, 10000)
(1000, 46)
test:
(2246, 10000)
(2246, 46)
```

TO DO: Define and train a network, then plot the accuracy of the training, validation, and testing

1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

In [96]:

```
reuters_model   = models.Sequential()
reuters_dropout = 0.2

reuters_model.add(layers.Dense(1024, activation="tanh", input_dim=10000))

reuters_model.add(layers.Dropout(reuters_dropout))

reuters_model.add(layers.Dense(256, activation="relu"))
reuters_model.add(layers.Dense(128, activation="relu"))

reuters_model.add(layers.Dropout(reuters_dropout))

reuters_model.add(layers.Dense(46, activation="softmax"))

reuters_model.compile(
    loss= "categorical_crossentropy",
    optimizer= "adamax",
    metrics= ["accuracy"]
)

reuters_model.summary()


reuters_early_stops = [
    callbacks.EarlyStopping(monitor= 'val_loss', patience= 4),
    callbacks.EarlyStopping(monitor= 'val_acc',  patience= 5)
]
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_51 (Dense)             (None, 1024)              10241024  
_________________________________________________________________
dropout_19 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_52 (Dense)             (None, 256)               262400    
_________________________________________________________________
dense_53 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_20 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_54 (Dense)             (None, 46)                5934      
=================================================================
Total params: 10,542,254
Trainable params: 10,542,254
Non-trainable params: 0
_________________________________________________________________
```

In [97]:

```python
reuters_model_history = reuters_model.fit(
    reuters_train_data,
    reuters_train_labels,
    batch_size= 1024,
    epochs= 16,
    verbose= 2,
    callbacks= reuters_early_stops,
    validation_data= (reuters_val_data, reuters_val_labels)
)
```

```
Train on 7982 samples, validate on 1000 samples
Epoch 1/16
 - 8s - loss: 2.5642 - acc: 0.4818 - val_loss: 1.5615 - val_acc: 0.656
0
Epoch 2/16
 - 7s - loss: 1.3029 - acc: 0.7129 - val_loss: 1.1722 - val_acc: 0.731
0
Epoch 3/16
 - 7s - loss: 0.9083 - acc: 0.7968 - val_loss: 0.9980 - val_acc: 0.789
0
Epoch 4/16
 - 7s - loss: 0.6427 - acc: 0.8513 - val_loss: 0.9031 - val_acc: 0.812
0
Epoch 5/16
 - 7s - loss: 0.4676 - acc: 0.8968 - val_loss: 0.8801 - val_acc: 0.822
0
Epoch 6/16
 - 7s - loss: 0.3395 - acc: 0.9266 - val_loss: 0.9141 - val_acc: 0.812
0
Epoch 7/16
 - 7s - loss: 0.2594 - acc: 0.9381 - val_loss: 0.9113 - val_acc: 0.821
0
Epoch 8/16
 - 7s - loss: 0.2066 - acc: 0.9461 - val_loss: 0.9337 - val_acc: 0.816
0
Epoch 9/16
 - 7s - loss: 0.1731 - acc: 0.9544 - val_loss: 0.9800 - val_acc: 0.818
0
```

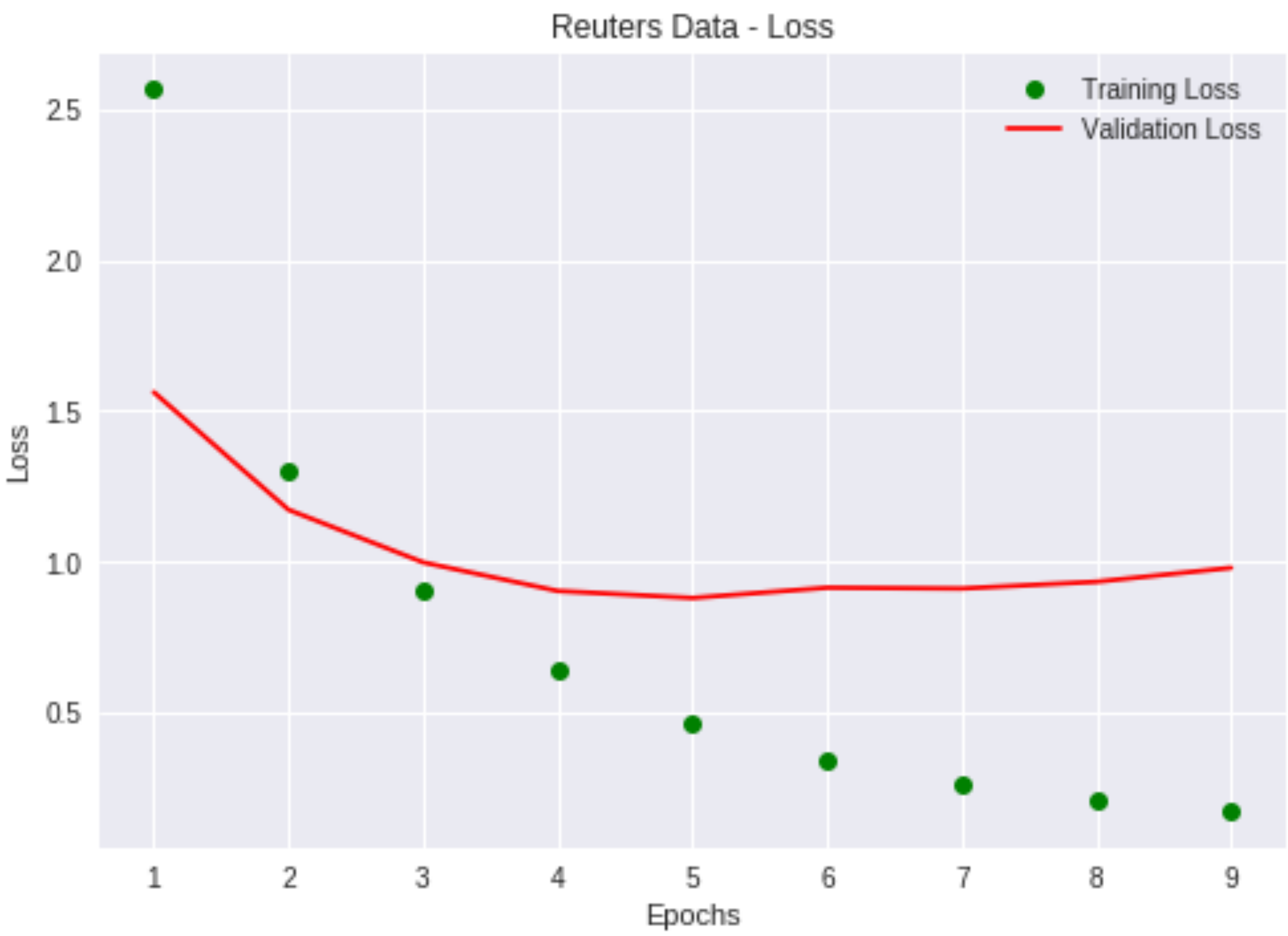
In [98]:

```
reuters_result = reuters_model.evaluate(
    reuters_test_data_token,
    reuters_one_hot_test_labels
)

print('Fashion score: {}%'.format(reuters_result[1]*100))
```

```
2246/2246 [==============================] - 2s 721us/step
Fashion score: 80.00890471950134%
```

In [100]:

```
reuters_history  = reuters_model_history.history
reuters_loss     = reuters_history['loss']
reuters_val_loss = reuters_history['val_loss']
reuters_epochs   = range(1, len(reuters_loss) + 1)

plt.plot(reuters_epochs, reuters_loss, 'go', label='Training Loss')
plt.plot(reuters_epochs, reuters_val_loss, 'r', label='Validation Loss')

plt.title('Reuters Data - Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [101]:

```
reuters_acc     = reuters_history['acc']
reuters_val_acc = reuters_history['val_acc']

plt.plot(reuters_epochs, reuters_acc, 'go', label='Training Acc')
plt.plot(reuters_epochs, reuters_val_acc, 'r', label='Validation Acc')

plt.title('Reuters Data - Accuracy')

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Reuters conclusion

> 1. This model was harder to achieve the accuracy goal. Like the past model, first, I over fitted the model but the accuracy in that point was at it's lowest in all the exercise.
> 2. Moving hyper parameters wasn't getting me closer to the goal. Neither changing my preprocessing work flow.
> 3. My final conclusion is that I couldn't get an final accuracy higher than 78% 'cause of the dataset values. My hypothesis is the data is not evenly given to me so I have more cases of 1 category than another.

Predicting Student Admissions

Predict student admissions based on three pieces of data:

- GRE Scores
- GPA Scores
- Class rank

Load and visualize the data

In [3]:

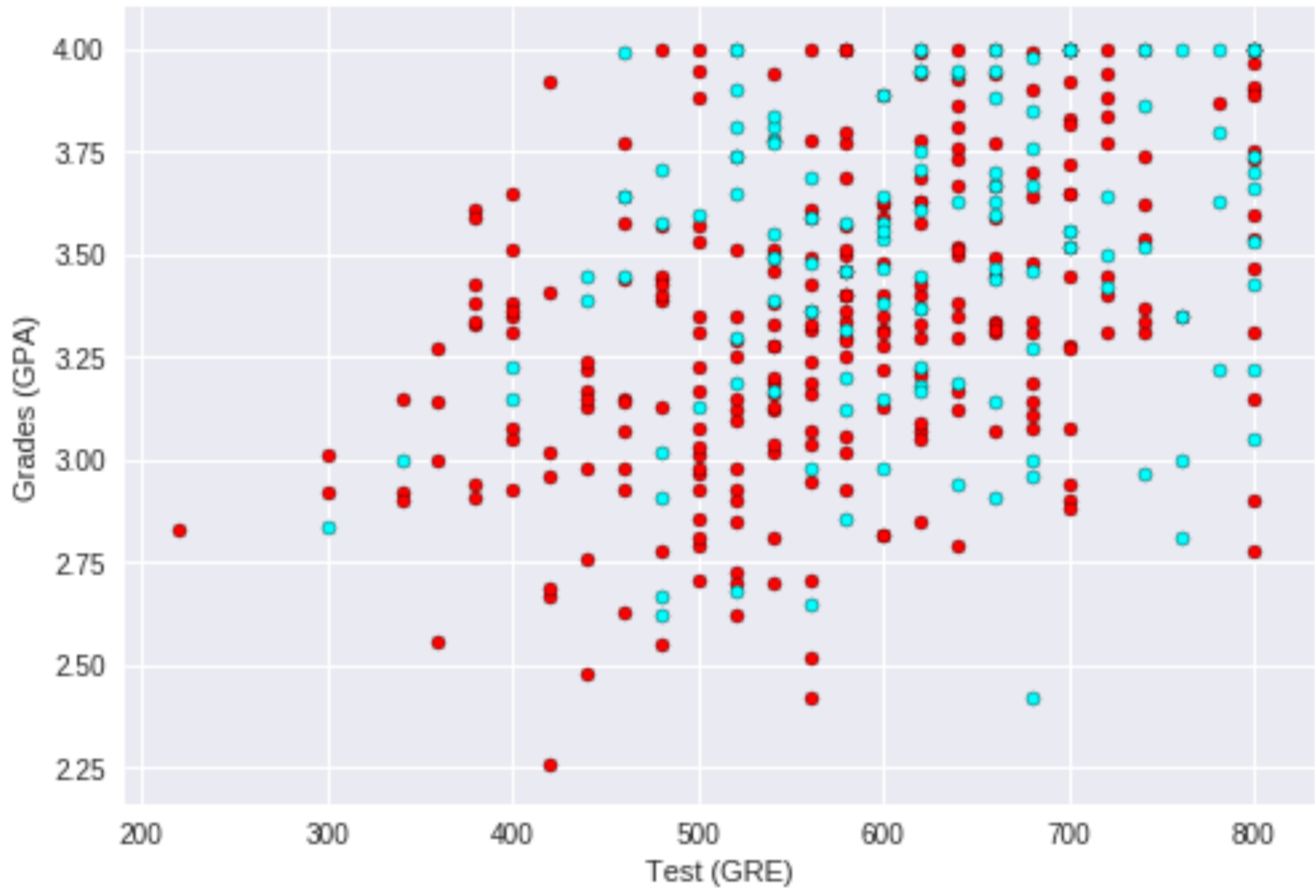
```
student_data = pd.read_csv("student_data.csv")
```

executed in 25ms, finished 13:44:56 2019-02-19

Plot of the GRE and the GPA from the data.

In [103]:

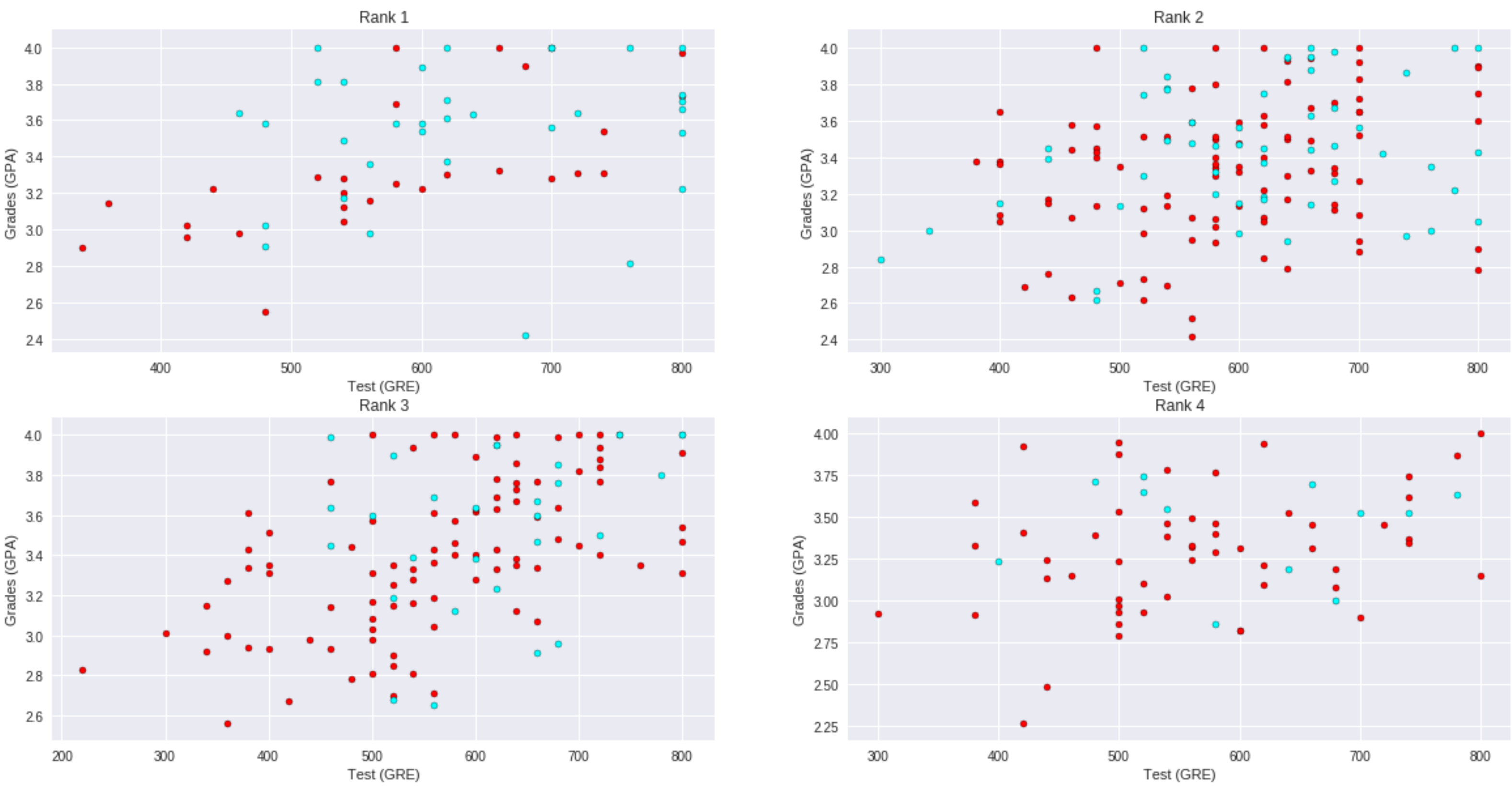
```
X = np.array(student_data[["gre","gpa"]])
y = np.array(student_data["admit"])
admitted = X[np.argwhere(y==1)]
rejected = X[np.argwhere(y==0)]
plt.scatter([s[0][0] for s in rejected], [s[0][1] for s in rejected], s = 25,
plt.scatter([s[0][0] for s in admitted], [s[0][1] for s in admitted], s = 25,
plt.xlabel('Test (GRE)')
plt.ylabel('Grades (GPA)')

plt.show()
```



Plot of the data by class rank.

In [104]:

```
f, plots = plt.subplots(2, 2, figsize=(20,10))
plots = [plot for sublist in plots for plot in sublist]

for idx, plot in enumerate(plots):
    data_rank = student_data[student_data["rank"]==idx+1]
    plot.set_title("Rank " + str(idx+1))
    X = np.array(data_rank[["gre","gpa"]])
    y = np.array(data_rank["admit"])
    admitted = X[np.argwhere(y==1)]
    rejected = X[np.argwhere(y==0)]
    plot.scatter([s[0][0] for s in rejected], [s[0][1] for s in rejected], s =
    plot.scatter([s[0][0] for s in admitted], [s[0][1] for s in admitted], s =
    plot.set_xlabel('Test (GRE)')
    plot.set_ylabel('Grades (GPA)')

```



TO DO: Preprocess the data

1. Normalize the input data set
2. Perform one hot encoding
3. Create a train, test, and validation set

In [6]:

```
# Replace nan with 0
student_data.fillna(value= 0, inplace= True)
# Shuffle the dataframe with pandas
student_data = student_data.sample(frac= 1).reset_index(drop= True)

# x
gre,   gpa  = np.array(student_data['gre']),   np.array(student_data['gpa'])
# y
admit, rank = np.array(student_data['admit']), np.array(student_data['rank'])

# Make everything in range from 0 - 1 (Normal distribution)
gpa = (gpa - gpa.mean(axis= 0)) / gpa.std(axis= 0)
gre = (gre - gre.mean(axis= 0)) / gre.std(axis= 0)

normalized_student_data = np.zeros((len(gpa), 2))
normalized_student_data[:,0], normalized_student_data[:,1] = gpa, gre

print(normalized_student_data.shape)

# one hot encoding
rank_one_hot = to_categorical(rank)

# train: 0 300, test: 300 350, val: 350 4000
student_train_data   = normalized_student_data[:300]
student_train_labels = rank_one_hot[:300]

student_test_data    = normalized_student_data[300:350]
student_test_labels  = rank_one_hot[300:350]

student_val_data     = normalized_student_data[350:]
student_val_labels   = rank_one_hot[350:]
```

executed in 12ms, finished 13:45:11 2019-02-19

```
(400, 2)
```

TO DO: Define and train a network, then plot the accuracy of the training, validation, and testing

1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

In [24]:

```
student_model = models.Sequential()
student_dropout = 0.3

student_model.add(layers.Dense(128, activation= 'sigmoid', input_shape=(2,)))

student_model.add(layers.Dropout(student_dropout))

student_model.add(layers.Dense(10, activation= 'sigmoid'))
student_model.add(layers.Dense(5,  activation= 'sigmoid'))

student_model.compile(
    optimizer= "rmsprop",
    loss= "binary_crossentropy",
    metrics=["accuracy"]
)

student_model.summary()
```

executed in 147ms, finished 13:50:34 2019-02-19

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 128)               384       
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290      
_________________________________________________________________
dense_15 (Dense)             (None, 5)                 55        
=================================================================
Total params: 1,729
Trainable params: 1,729
Non-trainable params: 0
_________________________________________________________________
```

In [25]:

```
student_model_history = student_model.fit(
  student_train_data,
  student_train_labels,
  epochs= 16,
  batch_size= 64,
  validation_data= (student_val_data, student_val_labels),
  verbose= 2
)
```

executed in 1.22s, finished 13:50:36 2019-02-19

```
Train on 300 samples, validate on 50 samples
Epoch 1/16
 - 0s - loss: 0.6999 - acc: 0.6373 - val_loss: 0.6536 - val_acc: 0.696
0
Epoch 2/16
 - 0s - loss: 0.6529 - acc: 0.6487 - val_loss: 0.6241 - val_acc: 0.696
0
Epoch 3/16
 - 0s - loss: 0.6258 - acc: 0.6507 - val_loss: 0.6031 - val_acc: 0.696
0
Epoch 4/16
 - 0s - loss: 0.6048 - acc: 0.6580 - val_loss: 0.5872 - val_acc: 0.696
0
Epoch 5/16
 - 0s - loss: 0.5900 - acc: 0.6647 - val_loss: 0.5744 - val_acc: 0.696
0
Epoch 6/16
 - 0s - loss: 0.5773 - acc: 0.6640 - val_loss: 0.5638 - val_acc: 0.696
0
Epoch 7/16
 - 0s - loss: 0.5650 - acc: 0.6713 - val_loss: 0.5548 - val_acc: 0.696
0
Epoch 8/16
 - 0s - loss: 0.5552 - acc: 0.7013 - val_loss: 0.5470 - val_acc: 0.764
0
Epoch 9/16
 - 0s - loss: 0.5478 - acc: 0.7560 - val_loss: 0.5402 - val_acc: 0.800
0
Epoch 10/16
 - 0s - loss: 0.5401 - acc: 0.7933 - val_loss: 0.5339 - val_acc: 0.800
0
Epoch 11/16
 - 0s - loss: 0.5335 - acc: 0.7993 - val_loss: 0.5282 - val_acc: 0.800
0
Epoch 12/16
 - 0s - loss: 0.5268 - acc: 0.7993 - val_loss: 0.5229 - val_acc: 0.800
0
Epoch 13/16
 - 0s - loss: 0.5213 - acc: 0.8000 - val_loss: 0.5179 - val_acc: 0.800
0
Epoch 14/16
 - 0s - loss: 0.5167 - acc: 0.8000 - val_loss: 0.5134 - val_acc: 0.800
0
Epoch 15/16
 - 0s - loss: 0.5114 - acc: 0.8000 - val_loss: 0.5091 - val_acc: 0.800
0
Epoch 16/16
 - 0s - loss: 0.5072 - acc: 0.8000 - val_loss: 0.5050 - val_acc: 0.800
0
```

In [26]:

```
student_result = student_model.evaluate(
  student_test_data,
  student_test_labels
)

print('Student score: {}'.format(student_result[1] * 100))
```

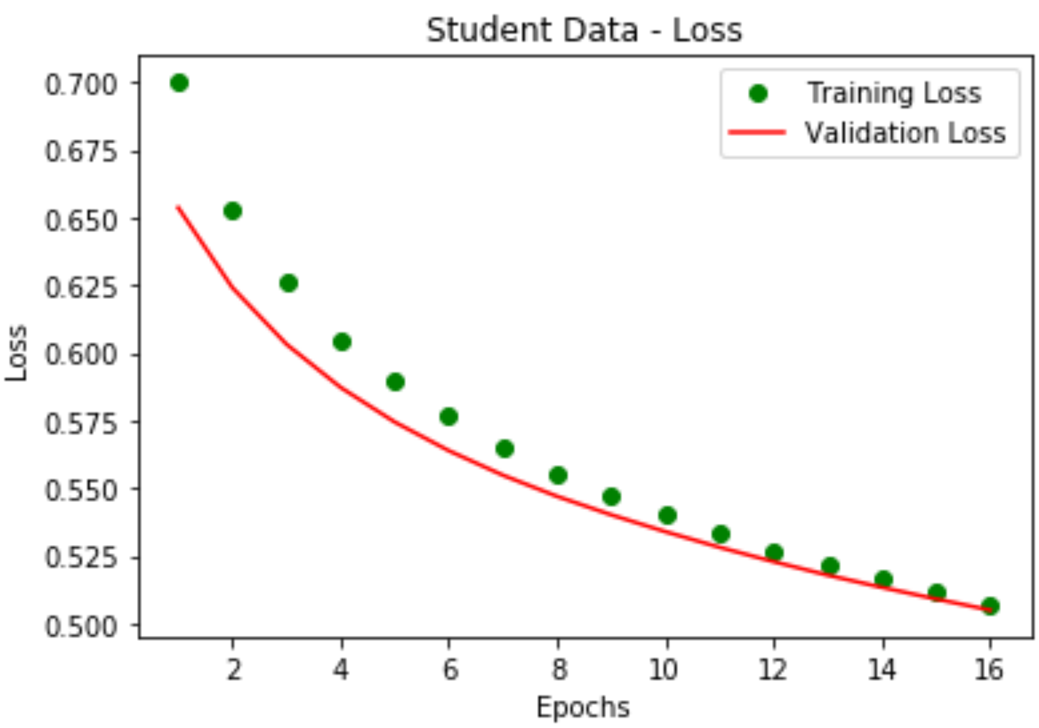
executed in 12ms, finished 13:50:41 2019-02-19

```
50/50 [==============================] - 0s 132us/step
Student score: 80.00000500679016
```

In [27]:

```
student_history = student_model_history.history

student_val_loss = student_history['val_loss']
student_loss     = student_history['loss']
student_epochs   = range(1, len(student_loss) + 1)

plt.plot(student_epochs, student_loss, 'go', label= "Training Loss")
plt.plot(student_epochs, student_val_loss, 'r', label= "Validation Loss")
plt.title("Student Data - Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
plt.show()
```

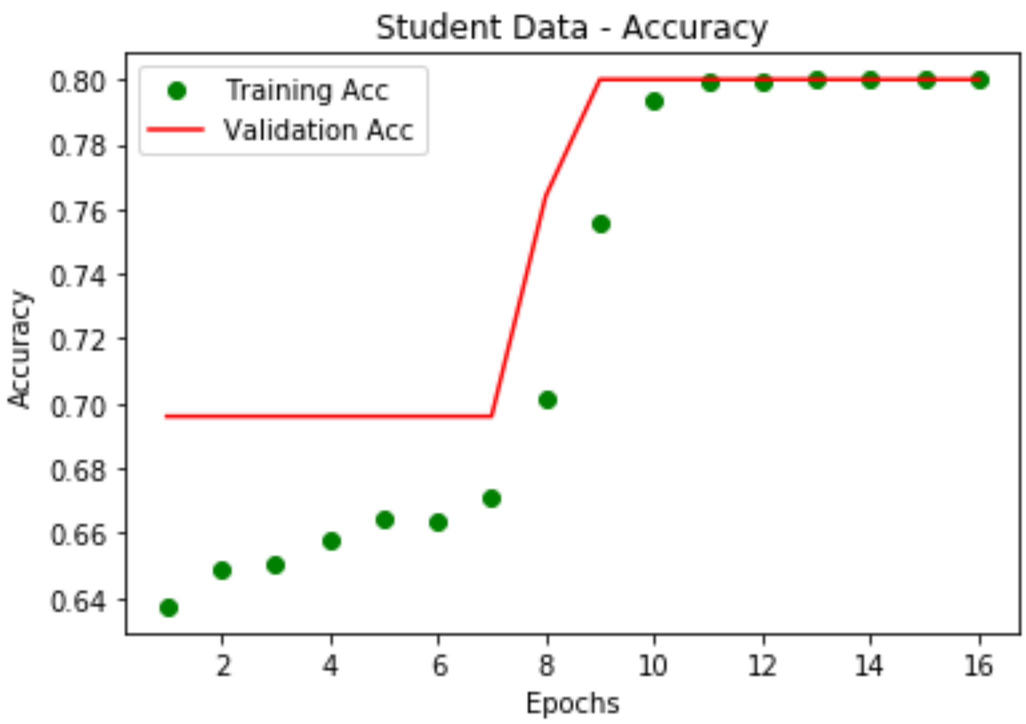
executed in 193ms, finished 13:50:43 2019-02-19



In [28]:

```python
student_val_acc = student_history['val_acc']
student_acc     = student_history['acc']

plt.plot(student_epochs, student_acc, 'go', label= "Training Acc")
plt.plot(student_epochs, student_val_acc, 'r', label= "Validation Acc")
plt.title("Student Data - Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
plt.legend()
plt.show()
```

executed in 197ms, finished 13:50:44 2019-02-19



Student conclusion

> This model was the hardest given the small dataset provided, we had 400 rows for training, validation & testing.
> I believe that my preprocessing work flow could be better due that I do not use any statistical function to better the data.

In [ ]:

```python

```