1 Classify different data sets

1.0.1 Basic includes

In [1]:

```
# 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
```

```
Using TensorFlow backend.
```

1.0.2 Classify the Fashion Mnist

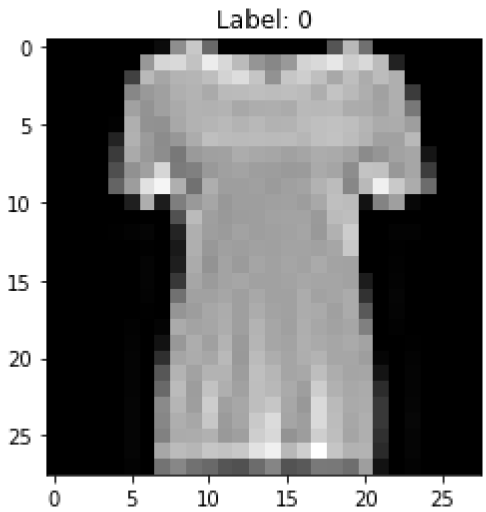
In [2]:

```
(fashion_train_data, fashion_train_labels), (fashion_test_data, fashion_test_

print(fashion_train_data.shape)

test_index = 10

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

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

Out[2]:

```
<matplotlib.image.AxesImage at 0x228df83b898>
```



1.0.2.1 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 [10]:

```
fashion_class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

fashion_train_data = fashion_train_data.reshape(60000, 28 * 28)
fashion_train_data = fashion_train_data.astype('float32') / 255
fashion_test_data = fashion_test_data.reshape(10000, 28 * 28)
fashion_test_data = fashion_test_data.astype('float32') / 255

fashion_train_labels = to_categorical(fashion_train_labels)
fashion_test_labels = to_categorical(fashion_test_labels)

fashion_validation_data = fashion_train_data[:5000]
fashion_validation_labels = fashion_train_labels[:5000]

fashion_x_data = fashion_train_data[5000:]
fashion_y_data = fashion_train_labels[5000:]
```

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

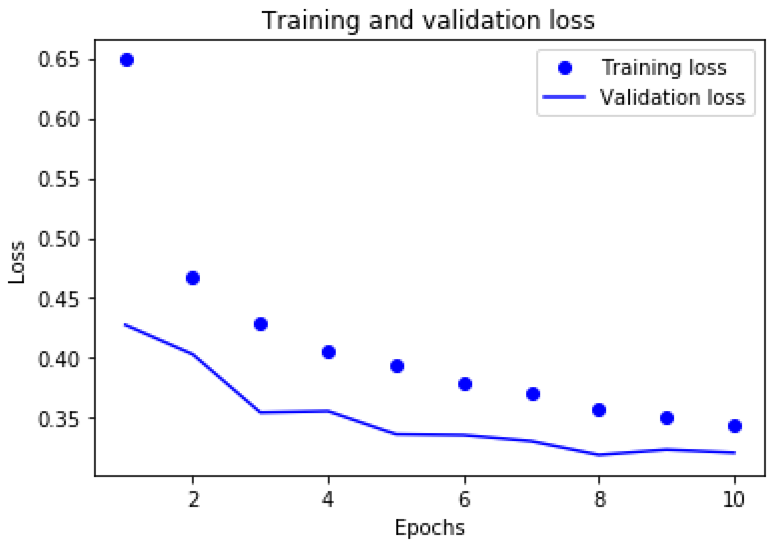
1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

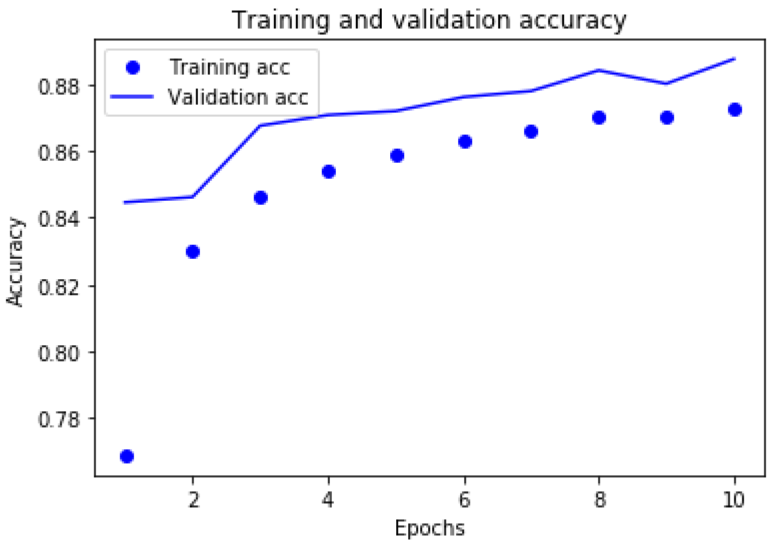
In [4]:

```
fashion_NN_model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(28 * 28,)),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(10, activation='softmax')
])

early_stop = callbacks.EarlyStopping(monitor='val_loss', patience=5)

fashion_NN_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

fashion_NN_model.summary()

fashion_history = fashion_NN_model.fit(fashion_x_data, fashion_y_data, epochs=

fashion_results = fashion_NN_model.evaluate(fashion_test_data, fashion_test_la

print('Fashion Test accuracy:', fashion_results)

# This dictionary stores the validation and accuracy of the model throughout t
fashion_history_dict = fashion_history.history
print(fashion_history_dict.keys())

# The history values are split in different lists for ease of plotting
fashion_acc = fashion_history_dict['acc']
fashion_val_acc = fashion_history_dict['val_acc']
fashion_loss = fashion_history_dict['loss']
fashion_val_loss = fashion_history_dict['val_loss']

fashion_epochs = range(1, len(fashion_acc) + 1)

# Plot of the validation and training loss

# "bo" is for "blue dot"
plt.plot(fashion_epochs, fashion_loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(fashion_epochs, fashion_val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Plot of the validation and train accuracy

plt.clf()   # clear figure

plt.plot(fashion_epochs, fashion_acc, 'bo', label='Training acc')
plt.plot(fashion_epochs, fashion_val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()

plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 128)               100480    
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650       
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
Train on 55000 samples, validate on 5000 samples
Epoch 1/10
 - 6s - loss: 0.6489 - acc: 0.7688 - val_loss: 0.4277 - val_acc: 0.8446
Epoch 2/10
 - 5s - loss: 0.4678 - acc: 0.8302 - val_loss: 0.4031 - val_acc: 0.8462
Epoch 3/10
 - 5s - loss: 0.4295 - acc: 0.8464 - val_loss: 0.3545 - val_acc: 0.8676
Epoch 4/10
 - 5s - loss: 0.4057 - acc: 0.8539 - val_loss: 0.3557 - val_acc: 0.8708
Epoch 5/10
 - 6s - loss: 0.3944 - acc: 0.8588 - val_loss: 0.3364 - val_acc: 0.8720
Epoch 6/10
 - 6s - loss: 0.3794 - acc: 0.8631 - val_loss: 0.3357 - val_acc: 0.8762
Epoch 7/10
 - 5s - loss: 0.3705 - acc: 0.8659 - val_loss: 0.3306 - val_acc: 0.8780
Epoch 8/10
 - 5s - loss: 0.3578 - acc: 0.8705 - val_loss: 0.3191 - val_acc: 0.8842
Epoch 9/10
 - 6s - loss: 0.3507 - acc: 0.8703 - val_loss: 0.3235 - val_acc: 0.8802
Epoch 10/10
 - 5s - loss: 0.3441 - acc: 0.8729 - val_loss: 0.3210 - val_acc: 0.8876
10000/10000 [==============================] - 0s 30us/step
Fashion Test accuracy: [0.3544481336593628, 0.8733]
dict_keys(['loss', 'val_acc', 'acc', 'val_loss'])
```





1.0.2.3 Fashion_comments

For this model, the tendency was to overfit and then adjust the parameters to improve the precision.

1.1 Classifying newswires

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

1.1.1 Load and review the data

In [16]:

```python
(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, 8
64, 39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 15, 7, 48, 9, 4579, 100
5, 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, 4
1, 42, 43, 44, 45}
```

Load the word index to decode the train data.

In [17]:

```python
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 acquisition of sp
ace co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share up
from 70 cts in 1986 the company said pretax net should rise to nine to 10 mln
dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to 22 mln d
lrs from 12 5 mln dlrs it said cash flow per share this year should be 2 50 to
three dlrs reuter 3
```

1.1.1.1 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 [18]:

```
tokenizer = Tokenizer(num_words=10000)
reuters_train_data_token = tokenizer.sequences_to_matrix(reuters_train_data, m
reuters_test_data_token = tokenizer.sequences_to_matrix(reuters_test_data, mod

# One-hot encoding the output
reuters_train_labels = to_categorical(reuters_train_labels, 46)
reuters_test_labels = to_categorical(reuters_test_labels, 46)

# Creating a validation set with the first 10000 reviews
reuters_validation_data = reuters_train_data_token[:1000]
reuters_validation_labels = reuters_train_labels[:1000]

# Creating the input set
reuters_x_data = reuters_train_data_token[1000:]
reuters_y_data = reuters_train_labels[1000:]
```

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

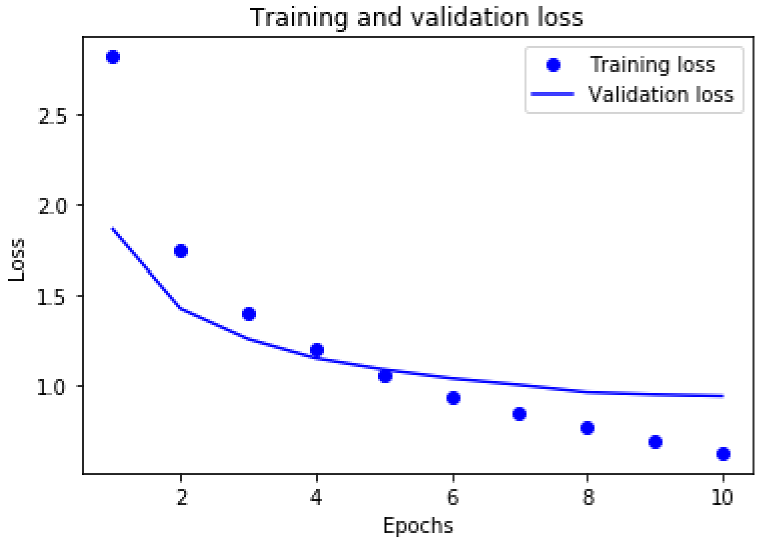
1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

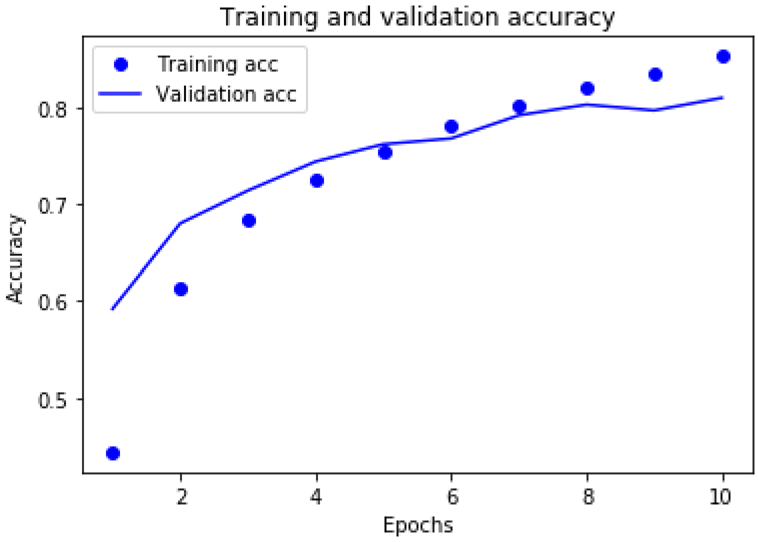
In [26]:

```
reuters_NN_model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(10000,)),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(46, activation='softmax')
    ])

reuters_NN_model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

reuters_NN_model.summary()

reuters_history = reuters_NN_model.fit(reuters_x_data, reuters_y_data, epochs=

reuters_results = reuters_NN_model.evaluate(reuters_test_data_token, reuters_t

print('Reuters Test accuracy:', reuters_results)

# This dictionary stores the validation and accuracy of the model throughout t
reuters_history_dict = reuters_history.history
print(reuters_history_dict.keys())

# The history values are split in different lists for ease of plotting
reuters_acc = reuters_history_dict['acc']
reuters_val_acc = reuters_history_dict['val_acc']
reuters_loss = reuters_history_dict['loss']
reuters_val_loss = reuters_history_dict['val_loss']

reuters_epochs = range(1, len(reuters_acc) + 1)

# Plot of the validation and training loss

# "bo" is for "blue dot"
plt.plot(reuters_epochs, reuters_loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(reuters_epochs, reuters_val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Plot of the validation and train accuracy

plt.clf()   # clear figure

plt.plot(reuters_epochs, reuters_acc, 'bo', label='Training acc')
plt.plot(reuters_epochs, reuters_val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

```

```
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_19 (Dense)             (None, 64)                640064    
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_20 (Dense)             (None, 64)                4160      
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_21 (Dense)             (None, 46)                2990      
=================================================================
Total params: 647,214
Trainable params: 647,214
Non-trainable params: 0
_________________________________________________________________
Train on 7982 samples, validate on 1000 samples
Epoch 1/10
7982/7982 [==============================] - 2s 238us/step - loss: 2.8189 - ac
c: 0.4435 - val_loss: 1.8619 - val_acc: 0.5920
Epoch 2/10
7982/7982 [==============================] - 1s 164us/step - loss: 1.7436 - ac
c: 0.6128 - val_loss: 1.4262 - val_acc: 0.6800
Epoch 3/10
7982/7982 [==============================] - 1s 176us/step - loss: 1.3972 - ac
c: 0.6848 - val_loss: 1.2578 - val_acc: 0.7140
Epoch 4/10
7982/7982 [==============================] - 2s 232us/step - loss: 1.2058 - ac
c: 0.7256 - val_loss: 1.1514 - val_acc: 0.7440
Epoch 5/10
7982/7982 [==============================] - 1s 178us/step - loss: 1.0590 - ac
c: 0.7533 - val_loss: 1.0889 - val_acc: 0.7620
Epoch 6/10
7982/7982 [==============================] - 1s 176us/step - loss: 0.9411 - ac
c: 0.7818 - val_loss: 1.0395 - val_acc: 0.7680
Epoch 7/10
7982/7982 [==============================] - 1s 185us/step - loss: 0.8457 - ac
c: 0.8028 - val_loss: 1.0035 - val_acc: 0.7920
Epoch 8/10
7982/7982 [==============================] - 1s 176us/step - loss: 0.7707 - ac
c: 0.8213 - val_loss: 0.9626 - val_acc: 0.8030
Epoch 9/10
7982/7982 [==============================] - 1s 165us/step - loss: 0.6957 - ac
c: 0.8356 - val_loss: 0.9495 - val_acc: 0.7970
Epoch 10/10
7982/7982 [==============================] - 1s 167us/step - loss: 0.6262 - ac
c: 0.8533 - val_loss: 0.9416 - val_acc: 0.8100
2246/2246 [==============================] - 0s 164us/step
Reuters Test accuracy: 0.784951024095819
dict_keys(['loss', 'val_acc', 'acc', 'val_loss'])
```





1.1.1.3 Reuters_comments

Using the preprocessing seen in classes and then tuning the parameters of the model, i couldn't get past 80 of accuracy. Even the testing tended to overfit.

1.2 Predicting Student Admissions

Predict student admissions based on three pieces of data:

- GRE Scores
- GPA Scores
- Class rank

1.2.1 Load and visualize the data

In [2]:

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

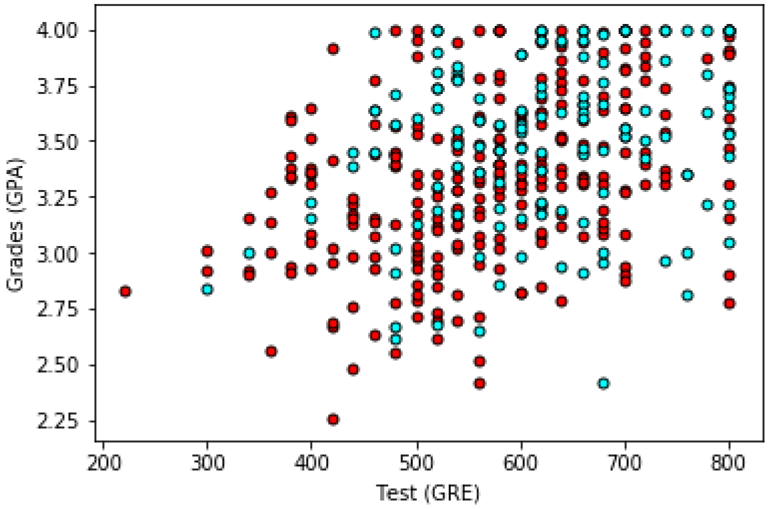
```
     admit    gre   gpa  rank
0        0  380.0  3.61   3.0
1        1  660.0  3.67   3.0
2        1  800.0  4.00   1.0
3        1  640.0  3.19   4.0
4        0  520.0  2.93   4.0
5        1  760.0  3.00   2.0
6        1  560.0  2.98   1.0
7        0  400.0  3.08   2.0
8        1  540.0  3.39   3.0
9        0  700.0  3.92   2.0
10       0  800.0  4.00   4.0
11       0  440.0  3.22   1.0
12       1  760.0  4.00   1.0
13       0  700.0  3.08   2.0
14       1  700.0  4.00   1.0
15       0  480.0  3.44   3.0
16       0  780.0  3.87   4.0
17       0  360.0  2.56   3.0
18       0  800.0  3.75   2.0
19       1  540.0  3.81   1.0
20       0  500.0  3.17   3.0
21       1  660.0  3.63   2.0
22       0  600.0  2.82   4.0
23       0  680.0  3.19   4.0
24       1  760.0  3.35   2.0
25       1  800.0  3.66   1.0
26       1  620.0  3.61   1.0
27       1  520.0  3.74   4.0
28       1  780.0  3.22   2.0
29       0  520.0  3.29   1.0
..     ...    ...   ...   ...
370      1  540.0  3.77   2.0
371      1  680.0  3.76   3.0
372      1  680.0  2.42   1.0
373      1  620.0  3.37   1.0
374      0  560.0  3.78   2.0
375      0  560.0  3.49   4.0
376      0  620.0  3.63   2.0
377      1  800.0  4.00   2.0
378      0  640.0  3.12   3.0
379      0  540.0  2.70   2.0
380      0  700.0  3.65   2.0
381      1  540.0  3.49   2.0
382      0  540.0  3.51   2.0
383      0  660.0  4.00   1.0
384      1  480.0  2.62   2.0
385      0  420.0  3.02   1.0
386      1  740.0  3.86   2.0
387      0  580.0  3.36   2.0
388      0  640.0  3.17   2.0
389      0  640.0  3.51   2.0
390      1  800.0  3.05   2.0
```

```
391       1  660.0  3.88   2.0
392       1  600.0  3.38   3.0
393       1  620.0  3.75   2.0
394       1  460.0  3.99   3.0
395       0  620.0  4.00   2.0
396       0  560.0  3.04   3.0
397       0  460.0  2.63   2.0
398       0  700.0  3.65   2.0
399       0  600.0  3.89   3.0

[400 rows x 4 columns]
```

Plot of the GRE and the GPA from the data.

In [9]:

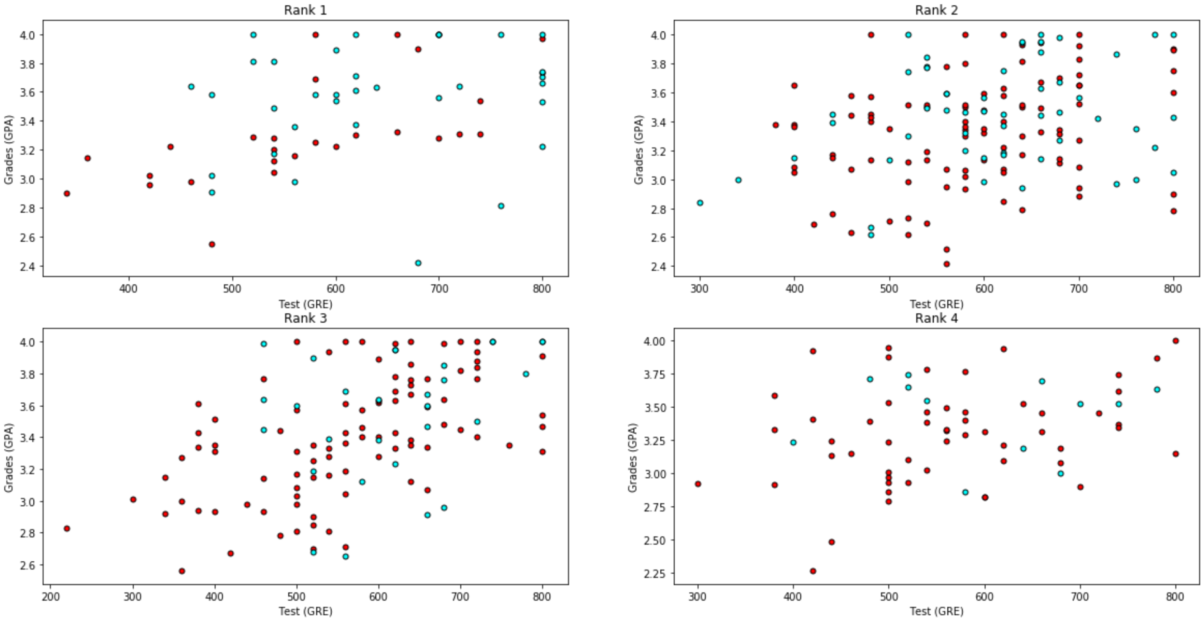
```
X = np.array(student_data[["gre","gpa"]])
y = np.array(student_data["admit"])
admitted = X[np.argwhere(y==1)]
rejected = X[np.argwhere(y==0)]
plt.scatter([s[0][0] for s in rejected], [s[0][1] for s in rejected], s = 25,
plt.scatter([s[0][0] for s in admitted], [s[0][1] for s in admitted], s = 25,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

plt.show()
```



Plot of the data by class rank.

In [10]:

```
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)')

```



1.2.1.1 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 [46]:

```python
student_data = student_data.fillna(0)

normalized_student_data = pd.get_dummies(student_data, columns=['rank'])
normalized_student_data["gre"] = normalized_student_data["gre"] / 800
normalized_student_data["gpa"] = normalized_student_data["gpa"] / 4

np.random.shuffle(normalized_student_data.values)

student_x = np.array(normalized_student_data)[:,1:]
student_x = student_x.astype('float32')
student_y = to_categorical(student_data["admit"])

# Creating a validation set
student_validation_data = student_x[:100]
student_validation_labels = student_y[:100]

# Creating the input set
student_x_data = student_x[100:]
student_y_data = student_y[100:]
```

1.2.1.2 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

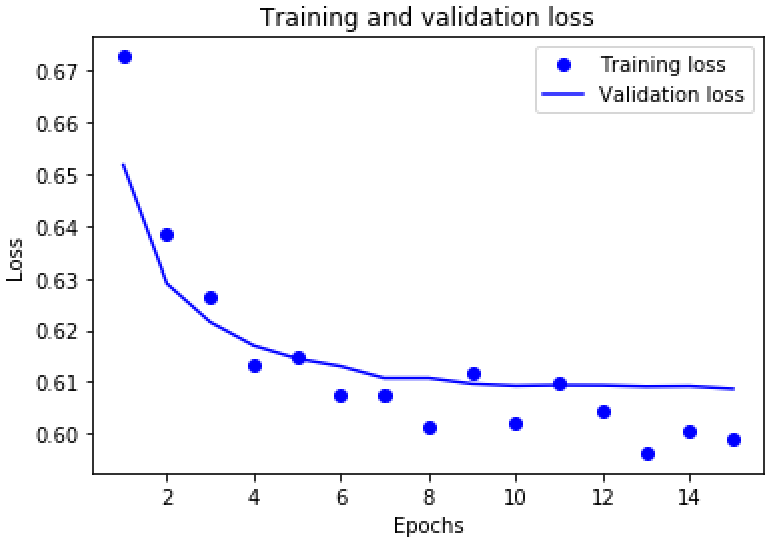
In [50]:

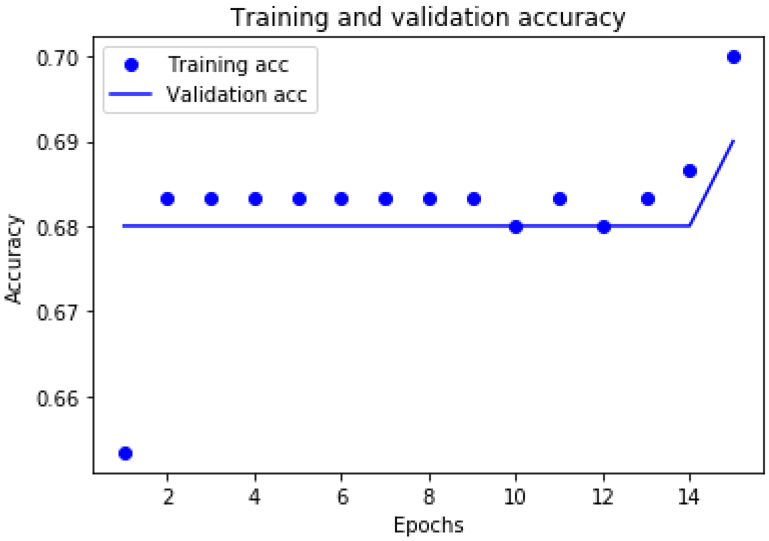
```
student_NN_model = models.Sequential([
    layers.Dense(128, activation='relu', kernel_initializer='random_uniform',
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(32, activation='relu'),
    layers.Dense(2, activation='softmax')
    ])

student_NN_model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

student_NN_model.summary()

student_history = student_NN_model.fit(student_x_data, student_y_data, epochs=

student_results = student_NN_model.evaluate(student_x[80:200], student_y[80:2

print('student Test accuracy:', student_results)

# This dictionary stores the validation and accuracy of the model throughout t
student_history_dict = student_history.history
print(student_history_dict.keys())

# The history values are split in different lists for ease of plotting
student_acc = student_history_dict['acc']
student_val_acc = student_history_dict['val_acc']
student_loss = student_history_dict['loss']
student_val_loss = student_history_dict['val_loss']

student_epochs = range(1, len(student_acc) + 1)

# Plot of the validation and training loss

# "bo" is for "blue dot"
plt.plot(student_epochs, student_loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(student_epochs, student_val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Plot of the validation and train accuracy

plt.clf()   # clear figure

plt.plot(student_epochs, student_acc, 'bo', label='Training acc')
plt.plot(student_epochs, student_val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```

plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_94 (Dense)             (None, 128)               1024
_________________________________________________________________
dropout_54 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_95 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_55 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_96 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_97 (Dense)             (None, 2)                 66
=================================================================
Total params: 11,426
Trainable params: 11,426
Non-trainable params: 0
_________________________________________________________________
Train on 300 samples, validate on 100 samples
Epoch 1/15
300/300 [==============================] - 1s 4ms/step - loss: 0.6728 - acc:
0.6533 - val_loss: 0.6518 - val_acc: 0.6800
Epoch 2/15
300/300 [==============================] - 0s 33us/step - loss: 0.6384 - acc:
0.6833 - val_loss: 0.6290 - val_acc: 0.6800
Epoch 3/15
300/300 [==============================] - 0s 43us/step - loss: 0.6265 - acc:
0.6833 - val_loss: 0.6216 - val_acc: 0.6800
Epoch 4/15
300/300 [==============================] - 0s 40us/step - loss: 0.6134 - acc:
0.6833 - val_loss: 0.6170 - val_acc: 0.6800
Epoch 5/15
300/300 [==============================] - 0s 47us/step - loss: 0.6149 - acc:
0.6833 - val_loss: 0.6145 - val_acc: 0.6800
Epoch 6/15
300/300 [==============================] - 0s 40us/step - loss: 0.6074 - acc:
0.6833 - val_loss: 0.6130 - val_acc: 0.6800
Epoch 7/15
300/300 [==============================] - 0s 43us/step - loss: 0.6075 - acc:
0.6833 - val_loss: 0.6107 - val_acc: 0.6800
Epoch 8/15
300/300 [==============================] - 0s 40us/step - loss: 0.6011 - acc:
0.6833 - val_loss: 0.6107 - val_acc: 0.6800
Epoch 9/15
300/300 [==============================] - 0s 47us/step - loss: 0.6119 - acc:
0.6833 - val_loss: 0.6096 - val_acc: 0.6800
Epoch 10/15
300/300 [==============================] - 0s 37us/step - loss: 0.6018 - acc:
0.6800 - val_loss: 0.6092 - val_acc: 0.6800
Epoch 11/15
300/300 [==============================] - 0s 47us/step - loss: 0.6099 - acc:
0.6833 - val_loss: 0.6094 - val_acc: 0.6800
```

```
Epoch 12/15
300/300 [==============================] - 0s 40us/step - loss: 0.6043 - acc:
0.6800 - val_loss: 0.6093 - val_acc: 0.6800
Epoch 13/15
300/300 [==============================] - 0s 47us/step - loss: 0.5963 - acc:
0.6833 - val_loss: 0.6091 - val_acc: 0.6800
Epoch 14/15
300/300 [==============================] - 0s 40us/step - loss: 0.6006 - acc:
0.6867 - val_loss: 0.6092 - val_acc: 0.6800
Epoch 15/15
300/300 [==============================] - 0s 50us/step - loss: 0.5989 - acc:
0.7000 - val_loss: 0.6086 - val_acc: 0.6900
120/120 [==============================] - 0s 50us/step
student Test accuracy: [0.5015921950340271, 0.800000003973643]
dict_keys(['loss', 'val_loss', 'acc', 'val_acc'])
```





1.2.1.3 Student_comments

The dataset was too small for my model to approach to a more trusty accuracy percentage, i had trouble in preprocessing and determining the size of the sets.