In [1]:
```
from sklearn.datasets import make_regression
# Below func. to generate dataset for Binary classification
from sklearn.datasets import make_circles
# Below func. tp generate dataset for Multi class classification
from sklearn.datasets import make_blobs
from numpy import where
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
# Import to_categorical to one hot encode the variables
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:
```
# generate regression dataset from a simple regression problem with a given
#number of input variables, statistical noise, and other properties
X, y = make_regression(n_samples = 1000, n_features = 20, noise = 0.1, random_state = 1)
# standardize dataset as both features and target variables have a gaussian distribution
X = StandardScaler().fit_transform(X)
y = StandardScaler().fit_transform(y.reshape(len(y),1))[:,0]
```

In [3]:
```
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

In [4]:
```
# Using a multi layer perceptron - 20 input nodes; 1 hidden layer with 25 nodes; 1 output node
# define model
model = Sequential()
model.add(Dense(25, input_dim = 20, activation = 'relu', kernel_initializer = 'he_uniform'))
model.add(Dense(1, activation='linear'))
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate
_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

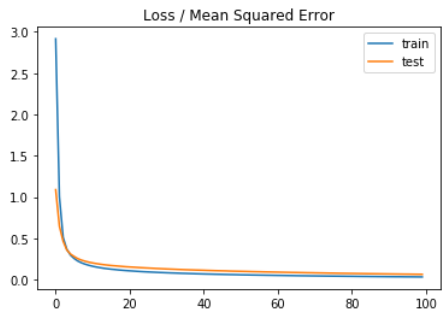
In [2]:
```
# Default learning rate = 0.01, Momentum = 0.9
SGD = optimizers.SGD(lr = 0.001, momentum = 0.9)
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate
_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Mean Squared Error Loss Function

In [6]:
```
# MSE mean_squared_error loss function
# MSE preferred loss function if target is gaussian distributed
model1 = model
model1.compile(loss = 'mean_squared_error', optimizer = SGD)
# fit model
history = model1.fit(trainX, trainy, validation_data = (testX, testy), epochs = 100, verbose = 0)
# Evaluate the model
train_mse = model1.evaluate(trainX, trainy, verbose = 0)
test_mse = model1.evaluate(testX, testy, verbose= 0)
print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training
plt.title('Loss / Mean Squared Error')
plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'test')
plt.legend()
plt.show()
```

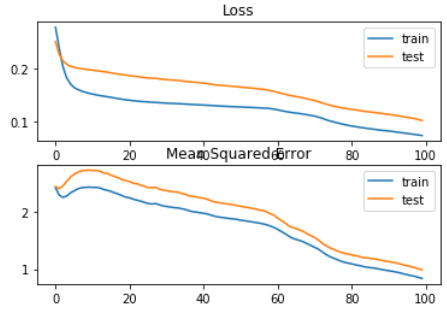
```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.034, Test: 0.066
```



Mean Squared Logarthmic Loss function

In [6]:
```
# MSLE loss function
# MSLE preferred if the target values are spread and when predicting a large value
# MSE punishes the model badly for large errors
# More appropriate to use when model predicts unscaled quantities directly

model2 = model
model2.compile(loss = 'mean_squared_logarithmic_error', optimizer = SGD, metrics = ['mse'])
# fit model
history = model2.fit(trainX, trainy, validation_data = (testX, testy), epochs = 100, verbose = 0)
# Evaluate the model
_, train_mse = model2.evaluate(trainX, trainy, verbose = 0)
_, test_mse = model2.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'test')
plt.legend()
# plot mse during training
plt.subplot(212)
plt.title('Mean Squared Error')
plt.plot(history.history['mean_squared_error'], label = 'train')
plt.plot(history.history['val_mean_squared_error'], label = 'test')
plt.legend()
plt.show()
```

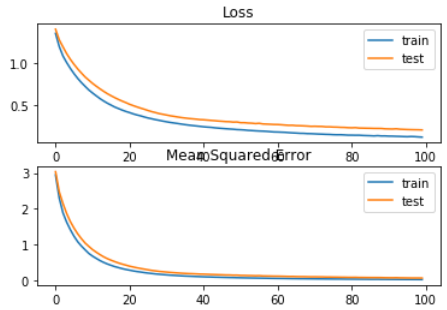
```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.827, Test: 0.990
```



Mean Absolute Error Loss function

In [6]:
```
# MAE Loss function
# MSLE not accurate as the distribution of the target variable is a standard Gaussian
# Appropriate loss function in case of outliers
# Calculated as the average of the absolute difference between the actual and predicted values

model3 = model
model.compile(loss = 'mean_absolute_error', optimizer = SGD, metrics = ['mse'])
# fit model
history = model3.fit(trainX, trainy, validation_data = (testX, testy), epochs = 100, verbose = 0)
# Evaluate the model
_, train_mse = model3.evaluate(trainX, trainy, verbose = 0)
_, test_mse = model3.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'test')
plt.legend()
# plot mse during training
plt.subplot(212)
plt.title('Mean Squared Error')
plt.plot(history.history['mean_squared_error'], label = 'train')
plt.plot(history.history['val_mean_squared_error'], label = 'test')
plt.legend()
plt.show()
```

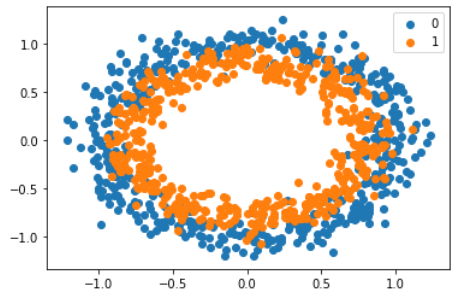
```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.031, Test: 0.073
```



In [ ]:
```
# MAE does converge but shows a bumpy course
# Here, target variable is a standard Gaussian with no large outliers, so MAE would not be a good fit in this case
```

Binary Classification Loss Functions

In [3]:
```
# scatter plot of the circles dataset with points colored by class
# generate circles
X, y = make_circles(n_samples = 1000, noise = 0.1, random_state = 1)
# select indices of points with each class label
for i in range(2):
    samples_ix = where(y == i)
    plt.scatter(X[samples_ix, 0], X[samples_ix, 1], label=str(i))
plt.legend()
plt.show()
```



In [4]:
```
# As points are already reasonably scaled around 0, almost in [-1,1], no rescaling in this case
```

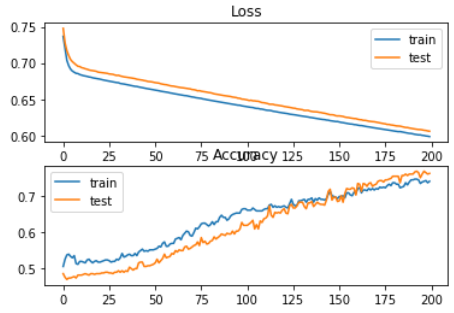
In [5]:
```
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

In [3]:
```
# define model
model = Sequential()
model.add(Dense(50, input_dim = 2, activation = 'relu', kernel_initializer = 'he_uniform'))
```

Binary Classification Cross Entropy

In [7]:
```
# Cross Entropy
# It calculate a score that summarizes the average difference between the actual
# and predicted probability distributions for predicting class 1
# The score is minimized and a perfect cross-entropy value is 0.

model1 = model

# Binary Cross Entropy requires that the output layer is configured with a single node
# and a 'sigmoid' activation in order to predict the probability for class 1

model1.add(Dense(1, activation='sigmoid'))

model1.compile(loss = 'binary_crossentropy', optimizer = SGD, metrics=['accuracy'])
# fit model
history = model1.fit(trainX, trainy, validation_data=(testX, testy), epochs = 200, verbose = 0)

#history_dict = history.history
#print(history_dict.keys())

# evaluate the model
_, train_acc = model1.evaluate(trainX, trainy, verbose = 0)
_, test_acc = model1.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'test')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['acc'], label = 'train')
plt.plot(history.history['val_acc'], label = 'test')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.742, Test: 0.764
```

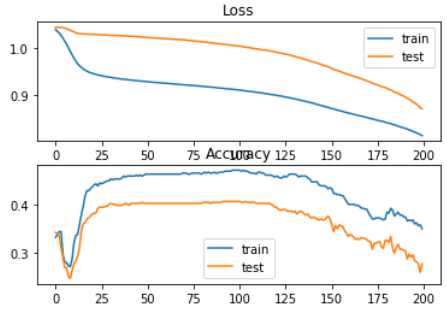


Binary Classification Hinge Loss

In [5]:
```
# Binary Classification Hinge Loss
# Intended for use with binary classification where the target values are in the set {-1, 1}.
# Target variable must be modified to have values in the set {-1, 1}
# change y from {0,1} to {-1,1}
y[where(y == 0)] = -1

# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

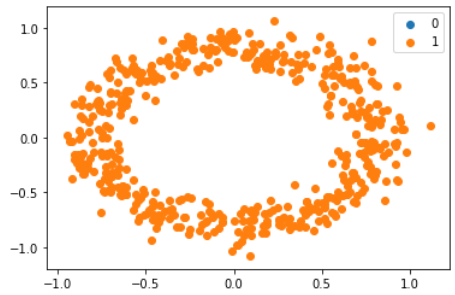
In [6]:
```
model2 = model
# Output layer of the network must be configured to have a single node with a hyperbolic tangent activation function
# capable of outputting a single value in the range [-1, 1]

model2.add(Dense(1, activation = 'tanh'))
model2.compile(loss = 'hinge', optimizer = SGD, metrics = ['accuracy'])
# fit model
history = model2.fit(trainX, trainy, validation_data=(testX, testy), epochs = 200, verbose = 0)
# evaluate the model
_, train_acc = model2.evaluate(trainX, trainy, verbose = 0)
_, test_acc = model2.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.350, Test: 0.278
```



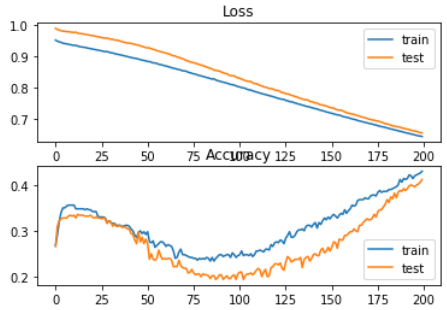
Binary Classification Squared Hinge Loss

In [4]:
```
# Binary Classification Squared Hinge Loss
# Intended for use with binary classification where the target values are in the set {-1, 1}.
# Target variable must be modified to have values in the set {-1, 1}
# scatter plot of the circles dataset with points colored by class
# generate circles

X, y = make_circles(n_samples = 1000, noise = 0.1, random_state = 1)
# select indices of points with each class label
# change y from {0,1} to {-1,1}
y[where(y == 0)] = -1
for i in range(2):
    samples_ix = where(y == i)
    plt.scatter(X[samples_ix, 0], X[samples_ix, 1], label=str(i))
plt.legend()
plt.show()

# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```



In [5]:
```
model3 = model
# Output layer of the network must be configured to have a single node with a hyperbolic tangent activation function
# capable of outputting a single value in the range [-1, 1]

model3.add(Dense(1, activation = 'tanh'))
model3.compile(loss = 'hinge', optimizer = SGD, metrics = ['accuracy'])
# fit model
history = model3.fit(trainX, trainy, validation_data=(testX, testy), epochs = 200, verbose = 0)
# evaluate the model
_, train_acc = model3.evaluate(trainX, trainy, verbose = 0)
_, test_acc = model3.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.430, Test: 0.412
```

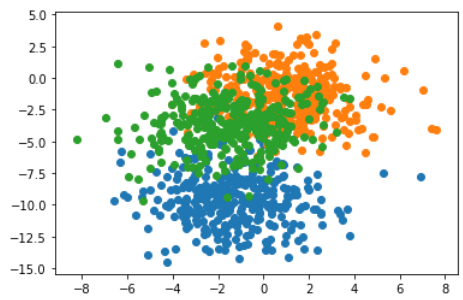


Multi-Class Classification Loss Functions

In [4]:
```
# generate dataset
# Use this function to generate 1,000 examples for a 3-class classification problem with 2 input variables
X, y = make_blobs(n_samples = 1000, centers = 3, n_features = 2, cluster_std = 2, random_state = 2)
```

Multi-class Categorical Cross Entropy

In [5]:
```
# scatter plot of blobs dataset
# select indices of points with each class label
for i in range(3):
    samples_ix = where(y == i)
    plt.scatter(X[samples_ix, 0], X[samples_ix, 1])
plt.show()
```

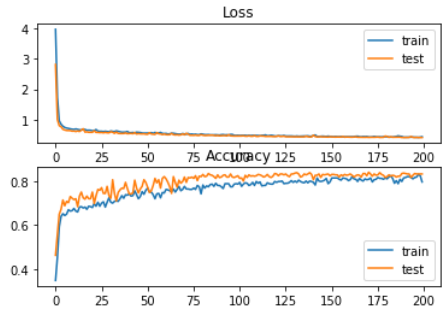


In [6]:
```
# To ensure that each example has an expected probability of 1.0 for the actual class value
# and an expected probability of 0.0 for all other class values, use 'to_categorical()' Keras function
# one hot encode output variable
y = to_categorical(y)
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

In [7]:
```
# Model expects two input variables (suppose same as before); Has 50 nodes in the hidden layer; Activation: Relu function;
```

In [8]:
```
model1 = model
model1.add(Dense(3, activation='softmax'))
model1.compile(loss = 'categorical_crossentropy', optimizer = SGD, metrics=['accuracy'])
# fit model
history = model1.fit(trainX, trainy, validation_data=(testX, testy), epochs = 200, verbose = 0)
# evaluate the model
_, train_acc = model1.evaluate(trainX, trainy, verbose = 0)
_, test_acc = model1.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.824, Test: 0.832
```

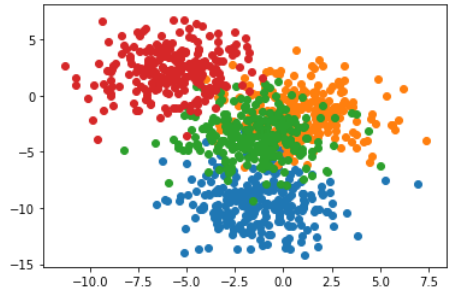


Sparse Multiclass Cross-Entropy Loss

In [ ]:
```
# Cross-Entropy has problems with a large number of labels during the one hot encoding process
# Ex.: Predicting words in a vocabulary may have tens or hundreds of thousands of categories, one for each label,
# This can mean that the target element of each training example may require a one hot encoded vector
# with tens or hundreds of thousands of zero values, requiring significant memory.
```

In [ ]:
```
# No requirement to One Hot encode the target variable
```

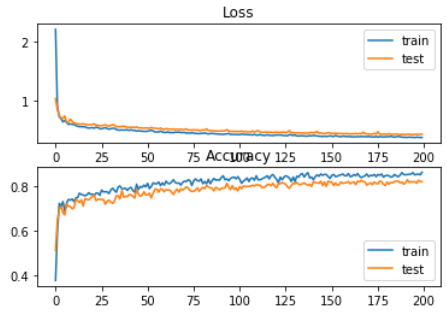
In [4]:
```
# generate dataset
# Use this function to generate 1,000 examples for a 4-class classification problem with 2 input variables
X, y = make_blobs(n_samples = 1000, centers = 4, n_features = 2, cluster_std = 2, random_state = 2)
# scatter plot of blobs dataset
# select indices of points with each class label
for i in range(4):
    samples_ix = where(y == i)
    plt.scatter(X[samples_ix, 0], X[samples_ix, 1])
plt.show()
```



In [5]:
```
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

In [6]:
```
model2 = model
model2.add(Dense(4, activation='softmax'))
model2.compile(loss = 'sparse_categorical_crossentropy', optimizer = SGD, metrics=['accuracy'])
# fit model
history = model2.fit(trainX, trainy, validation_data=(testX, testy), epochs = 200, verbose = 0)
# evaluate the model
_, train_acc = model2.evaluate(trainX, trainy, verbose = 0)
_, test_acc = model2.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.860, Test: 0.822
```

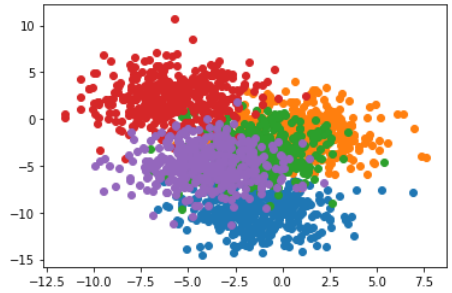


Kullback Leibler Divergence Loss

In [ ]:
```
# KL Divergence for short, is a measure of how one probability distribution differs from a baseline distribution
```

In [ ]:
```
# KL divergence loss function is more commonly used when using models that learn to approximate a more
# complex function than simply multi-class classification, such as in the case of an autoencoder used for
# learning a dense feature representation under a model that must reconstruct the original input.
# In this case, KL divergence loss would be preferred
```

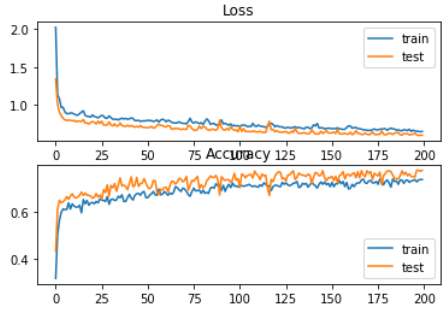
In [4]:
```
# generate dataset
# Use this function to generate 1,000 examples for a 5-class classification problem with 2 input variables
X, y = make_blobs(n_samples = 2000, centers = 5, n_features = 2, cluster_std = 2, random_state = 2)
# scatter plot of blobs dataset
# select indices of points with each class label
for i in range(5):
    samples_ix = where(y == i)
    plt.scatter(X[samples_ix, 0], X[samples_ix, 1])
plt.show()
```



In [5]:
```
# To ensure that each example has an expected probability of 1.0 for the actual class value
# and an expected probability of 0.0 for all other class values, use 'to_categorical()' Keras function
# one hot encode output variable
y = to_categorical(y)
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

In [6]:
```
model3 = model
model3.add(Dense(5, activation='softmax'))
model3.compile(loss = 'kullback_leibler_divergence', optimizer = SGD, metrics=['accuracy'])
# fit model
history = model3.fit(trainX, trainy, validation_data=(testX, testy), epochs = 200, verbose = 0)
# evaluate the model
_, train_acc = model3.evaluate(trainX, trainy, verbose = 0)
_, test_acc = model3.evaluate(testX, testy, verbose = 0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\deepai\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tens
orflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train: 0.740, Test: 0.773
```



In [ ]:
```
```