In [1]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.pipeline import Pipeline
import pandas as pd
import numpy as np
import pickle

""
```

```
C:\Users\Chuck B\Anaconda3\envs\tensorflow\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWar
ning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the
refactored classes and functions are moved. Also note that the interface of the new CV iterators are differ
ent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Load and Divide Data into Training and Testing

In addition, the data is scaled.

In [5]:
```
# Load comma delimited sensor data into Pandas data frame
sensor_file = "../data/sensor_11_12_2017_10_52_24.csv"
df = pd.read_csv(sensor_file)
print (df.shape)

# Extract features and target (y)
X = df.iloc[:,1:15] # exclude data
y = df.iloc[:,16]
print (X.shape)
print (y.shape)

# Use 80% of data for training, 20% for testing
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20, random_state = 1)
print (X_train.shape)
print (X_test.shape)

# Scaling features, mean = 0, stddev = 1
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)

# Head
df.head()

""
```

```
(5457, 17)
(5457, 14)
(5457,)
(4365, 14)
(1092, 14)
```

Out[5]:

	DateTime	Temp	Humidity	Pressure	Yaw	Pitch	Roll	MagX	MagY	MagZ	AccX	AccY	AccZ	
0	11/12/2017 10:52:35.251176	29.361	40.702	1031.956	215.801975	356.730118	2.171165	-5.351357	5.945352	26.308798	0.056247	0.034719	0.960166	-0.0
1	11/12/2017 10:52:35.523388	29.308	39.749	1031.952	216.099403	356.626506	2.103062	-10.720130	12.090914	53.852757	0.055035	0.016267	0.974544	0.0
2	11/12/2017 10:52:35.688238	29.468	40.231	1031.941	216.340904	356.301112	2.310486	-13.004756	14.781075	65.216164	0.046307	0.014810	0.939451	0.0
3	11/12/2017 10:52:35.849093	29.397	40.121	1031.950	216.613607	356.519507	2.298305	-13.880388	16.216396	69.694740	0.060854	0.016267	0.945544	-0.0
4	11/12/2017 10:52:36.010403	29.343	40.088	1031.943	216.940253	356.745464	2.196436	-13.895806	16.390114	71.753525	0.055763	0.023308	0.937014	-0.0

Construct Neural Network Model and Tweak

In [24]:
```
# Supervised neural network model
# http://scikit-learn.org/stable/modules/neural_networks_supervised.html

# NN has 3 hidden layers. 1st layer has 15 neurons which equals 15 features
#     2nd layer has 100 (just because), and 3rd layer has 8 which equals 8 classes

# Pipeline, push scaler and classifer
pipe = Pipeline([('scl', StandardScaler()),
                 ('clf', MLPClassifier(solver='lbfgs', alpha=0.00001,
                                       hidden_layer_sizes=(15,100, 8), random_state=1))])

# How well does the model converge? Overfitting? Underfitting?
scores = cross_val_score(estimator=pipe, X=X_train, y=y_train,cv=10,n_jobs=1)
for s in scores:
    print (s)

# The final CV acc for training data was 0.64 using Logistic Regression
print ('CV Acc: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))

""
```

```
0.909090909091
0.915717539863
0.913242009132
0.926940639269
0.91095890411
0.924137931034
0.885057471264
0.926267281106
0.917050691244
0.921658986175
CV Acc: 0.915 +/- 0.012
```

Use Models to Make Predictions

In [47]:
```
# Fit training data
pipe.fit(X_train, y_train)
predictions = pipe.predict(scaler.transform(X_test)) # Get predictions

""
```

Evaluate Model Performance on Testing Data

In [60]:
```
# Better table
pd.crosstab(y_test, predictions)

# Ideally, there would zeros everywhere except for the main diagonal
# Last row, for example, the actual state is 'walking', but the model
# is predicting 'turn-right'

""
```

Out[60]:

col_0 / State	go_down	go_up	spin_left	spin_right	standing	turn_left	turn_right	walking
go_down	29	9	2	0	4	0	3	2
go_up	5	41	2	2	0	0	1	2
spin_left	0	1	100	0	0	3	0	0
spin_right	0	0	0	66	1	0	0	0
standing	0	2	0	0	123	2	0	3
turn_left	1	1	0	0	4	159	3	9
turn_right	0	0	0	2	1	3	138	13
walking	0	1	0	0	3	5	10	336

In [28]:
```
# Metrics
precision, recall, fscore, support = score(y_test, predictions)

print ('Prec ','Rec','  Fscore')
for i in range(len(precision)):
    print (('%.3f' % precision[i]), ('%.3f' % recall[i]), ('%.3f' % fscore[i]))

## Note that accuracy is not appropriate for unbalanced classes.
## F-score is the superior metric
## The lowest Fscores are associated with 'go_down' and 'go_up'

""
```

```
Prec  Rec   Fscore
0.829 0.592 0.690
0.745 0.774 0.759
0.962 0.962 0.962
0.943 0.985 0.964
0.904 0.946 0.925
0.924 0.898 0.911
0.890 0.879 0.885
0.921 0.946 0.933
```

Deploying Demonstration

The following code demonstrate how the Neural Network model is saved and deployed to automatically classify sensor readings into one of eight classes.

I intend on loading this model to the Raspberry Pi and feeding it with the sensor data to indicate automatically what motion the user is making.

In [30]:
```
# Use pickle library to save model
filename = '../model/sensor_model_2017.11.14a.sav'
pickle.dump(pipe, open(filename, 'wb'))
```

In [46]:
```
# NOTE: For reference only
#Add this to data logger for automatically describing the state of motion

# Load our saved model
loaded_model = pickle.load(open(filename, 'rb'))

# Simulate 10 consecutive sensor readings and make a prediction
for i in range(100,110):
    prediction = loaded_model.predict(scaler.transform(X_test.iloc[i,:].reshape(1,-1)))
    print (prediction)


## It may be necessary to calculate a running average on sensors to minimize
## noise.

""
```

```
['walking']
['walking']
['turn_left']
['spin_left']
['turn_right']
['spin_right']
['spin_right']
['go_up']
['standing']
['walking']
```

```
C:\Users\Chuck B\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:9: FutureWarning: reshap
e is deprecated and will raise in a subsequent release. Please use .values.reshape(...) instead
  if __name__ == '__main__':
```

Compares Before and After Reshaping

In [43]:
```
# Reference only. In the next cell the data must be reshaped. Here are the differences.
print (X_test.iloc[1,:])
print (X_test.iloc[1,:].reshape(1,-1))
```

```
""
Temp          25.527000
Humidity      36.760000
Pressure    1025.950000
Yaw          158.921194
Pitch        356.681997
Roll           2.092825
MagX         -42.919319
MagY         -14.954748
MagZ          76.645782
AccX           0.042428
AccY           0.022094
AccZ           0.943594
GyroX          0.000503
GyroY          0.017684
Name: 3196, dtype: float64
[[  2.55270000e+01   3.67600000e+01   1.02595000e+03   1.58921194e+02
    3.56681997e+02   2.09282500e+00  -4.29193190e+01  -1.49547480e+01
    7.66457820e+01   4.24280000e-02   2.20940000e-02   9.43594000e-01
    5.03000000e-04   1.76840000e-02]]

C:\Users\Chuck B\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: reshap
e is deprecated and will raise in a subsequent release. Please use .values.reshape(...) instead
```