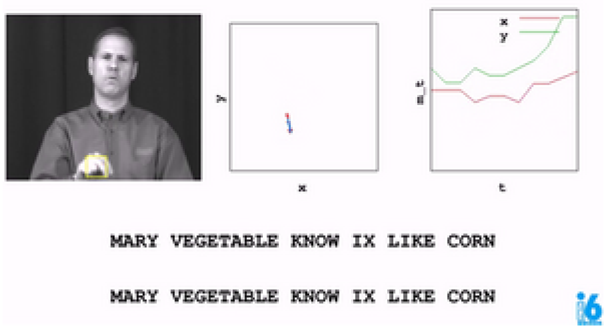
Artificial Intelligence Engineer Nanodegree - Probabilistic Models

Project: Sign Language Recognition System

Introduction

The overall goal of this project is to build a word recognizer for American Sign Language video sequences, demonstrating the power of probabalistic models. In particular, this project employs hidden Markov models (HMM's) (https://en.wikipedia.org/wiki/Hidden_Markov_model) to analyze a series of measurements taken from videos of American Sign Language (ASL) collected for research (see the RWTH-BOSTON-104 Database (http://www-i6.informatik.rwth-aachen.de/~dreuw/database-rwth-boston-104.php)). In this video, the right-hand x and y locations are plotted as the speaker signs the sentence.



(https://drive.google.com/open?id=0B_5qGuFe-wbhUXRuVnNZVnMtam8)

The raw data, train, and test sets are pre-defined. You will derive a variety of feature sets (explored in Part 1), as well as implement three different model selection criterion to determine the optimal number of hidden states for each word model (explored in Part 2). Finally, in Part 3 you will implement the recognizer and compare the effects the different combinations of feature sets and model selection criteria.

At the end of each Part, complete the submission cells with implementations, answer all questions, and pass the unit tests. Then submit the completed notebook for review!

PART 1: Data

Features Tutorial

Load the initial database

A data handler designed for this database is provided in the student codebase as the `AslDb` class in the `asl_data` module. This handler creates the initial pandas (http://pandas.pydata.org/pandas-docs/stable/) dataframe from the corpus of data included in the `data` directory as well as dictionaries suitable for extracting data in a format friendly to the hmmlearn (https://hmmlearn.readthedocs.io/en/latest/) library. We'll use those to create models in Part 2.

To start, let's set up the initial database and select an example set of features for the training set. At the end of Part 1, you will create additional feature sets for experimentation.

In [1]:
```
import numpy as np
import pandas as pd
from asl_data import AslDb


asl = AslDb() # initializes the database
asl.df.head() # displays the first five rows of the asl database, indexed by video and frame
```

Out[1]:

		left-x	left-y	right-x	right-y	nose-x	nose-y	speaker
video	**frame**							
98	**0**	149	181	170	175	161	62	woman-1
	1	149	181	170	175	161	62	woman-1
	2	149	181	170	175	161	62	woman-1
	3	149	181	170	175	161	62	woman-1
	4	149	181	170	175	161	62	woman-1

In [2]:
```
asl.df.ix[98,1]  # look at the data available for an individual frame
```

Out[2]:
```
left-x         149
left-y         181
right-x        170
right-y        175
nose-x         161
nose-y          62
speaker    woman-1
Name: (98, 1), dtype: object
```

The frame represented by video 98, frame 1 is shown here:

Feature selection for training the model

The objective of feature selection when training a model is to choose the most relevant variables while keeping the model as simple as possible, thus reducing training time. We can use the raw features already provided or derive our own and add columns to the pandas dataframe `asl.df` for selection. As an example, in the next cell a feature named `'grnd-ry'` is added. This feature is the difference between the right-hand y value and the nose y value, which serves as the "ground" right y value.

In [3]:
```
asl.df['grnd-ry'] = asl.df['right-y'] - asl.df['nose-y']
asl.df.head()  # the new feature 'grnd-ry' is now in the frames dictionary
```

Out[3]:

		left-x	left-y	right-x	right-y	nose-x	nose-y	speaker	grnd-ry
video	**frame**								
98	**0**	149	181	170	175	161	62	woman-1	113
	1	149	181	170	175	161	62	woman-1	113
	2	149	181	170	175	161	62	woman-1	113
	3	149	181	170	175	161	62	woman-1	113
	4	149	181	170	175	161	62	woman-1	113

Try it!

In [4]:
```
from asl_utils import test_features_tryit
# TODO add df columns for 'grnd-rx', 'grnd-ly', 'grnd-lx' representing differences between hand and nose loca
tions
asl.df['grnd-rx'] = asl.df['right-x'] - asl.df['nose-x']
asl.df['grnd-ly'] = asl.df['left-y'] - asl.df['nose-y']
asl.df['grnd-lx'] = asl.df['left-x'] - asl.df['nose-x']
# test the code
test_features_tryit(asl)
```

asl.df sample

		left-x	left-y	right-x	right-y	nose-x	nose-y	speaker	grnd-ry	grnd-rx	grnd-ly	grnd-lx
video	**frame**											
98	**0**	149	181	170	175	161	62	woman-1	113	9	119	-12
	1	149	181	170	175	161	62	woman-1	113	9	119	-12
	2	149	181	170	175	161	62	woman-1	113	9	119	-12
	3	149	181	170	175	161	62	woman-1	113	9	119	-12
	4	149	181	170	175	161	62	woman-1	113	9	119	-12

Out[4]: Correct!

In [5]:
```
# collect the features into a list
features_ground = ['grnd-rx','grnd-ry','grnd-lx','grnd-ly']
 #show a single set of features for a given (video, frame) tuple
[asl.df.ix[98,1][v] for v in features_ground]
print(asl.df.size)
print(len(asl.df))
print(len(asl.df.ix[98,1]))
print(asl.df.ix[1,8])
```

```
173206
15746
11
left-x         151
left-y         177
right-x        164
right-y        132
nose-x         160
nose-y          56
speaker    woman-1
grnd-ry         76
grnd-rx          4
grnd-ly        121
grnd-lx         -9
Name: (1, 8), dtype: object
```

Build the training set

Now that we have a feature list defined, we can pass that list to the `build_training` method to collect the features for all the words in the training set. Each word in the training set has multiple examples from various videos. Below we can see the unique words that have been loaded into the training set:

In [6]:
```
training = asl.build_training(features_ground)
print("Training words: {}".format(training.words))
```

```
Training words: ['JOHN', 'WRITE', 'HOMEWORK', 'IX-1P', 'SEE', 'YESTERDAY', 'IX', 'LOVE', 'MARY', 'CAN', 'G
O', 'GO1', 'FUTURE', 'GO2', 'PARTY', 'FUTURE1', 'HIT', 'BLAME', 'FRED', 'FISH', 'WONT', 'EAT', 'BUT', 'CHICK
EN', 'VEGETABLE', 'CHINA', 'PEOPLE', 'PREFER', 'BROCCOLI', 'LIKE', 'LEAVE', 'SAY', 'BUY', 'HOUSE', 'KNOW',
 'CORN', 'CORN1', 'THINK', 'NOT', 'PAST', 'LIVE', 'CHICAGO', 'CAR', 'SHOULD', 'DECIDE', 'VISIT', 'MOVIE', 'W
ANT', 'SELL', 'TOMORROW', 'NEXT-WEEK', 'NEW-YORK', 'LAST-WEEK', 'WILL', 'FINISH', 'ANN', 'READ', 'BOOK', 'CH
OCOLATE', 'FIND', 'SOMETHING-ONE', 'POSS', 'BROTHER', 'ARRIVE', 'HERE', 'GIVE', 'MAN', 'NEW', 'COAT', 'WOMA
N', 'GIVE1', 'HAVE', 'FRANK', 'BREAK-DOWN', 'SEARCH-FOR', 'WHO', 'WHAT', 'LEG', 'FRIEND', 'CANDY', 'BLUE',
 'SUE', 'BUY1', 'STOLEN', 'OLD', 'STUDENT', 'VIDEOTAPE', 'BORROW', 'MOTHER', 'POTATO', 'TELL', 'BILL', 'THRO
W', 'APPLE', 'NAME', 'SHOOT', 'SAY-1P', 'SELF', 'GROUP', 'JANA', 'TOY1', 'MANY', 'TOY', 'ALL', 'BOY', 'TEACH
ER', 'GIRL', 'BOX', 'GIVE2', 'GIVE3', 'GET', 'PUTASIDE']
```

The training data in `training` is an object of class `WordsData` defined in the `asl_data` module. in addition to the `words` list, data can be accessed with the `get_all_sequences`, `get_all_Xlengths`, `get_word_sequences`, and `get_word_Xlengths` methods. We need the `get_word_Xlengths` method to train multiple sequences with the `hmmlearn` library. In the following example, notice that there are two lists; the first is a concatenation of all the sequences(the X portion) and the second is a list of the sequence lengths(the Lengths portion).

In [7]:
```
training.get_word_Xlengths('CHOCOLATE')
```

Out[7]:
```
(array([[-11,  48,   7, 120],
        [-11,  48,   8, 109],
        [ -8,  49,  11,  98],
        [ -7,  50,   7,  87],
        [ -4,  54,   7,  77],
        [ -4,  54,   6,  69],
        [ -4,  54,   6,  69],
        [-13,  52,   6,  69],
        [-13,  52,   6,  69],
        [ -8,  51,   6,  69],
        [ -8,  51,   6,  69],
        [ -8,  51,   6,  69],
        [ -8,  51,   6,  69],
        [ -8,  51,   6,  69],
        [-10,  59,   7,  71],
        [-15,  64,   9,  77],
        [-17,  75,  13,  81],
        [ -4,  48,  -4, 113],
        [ -2,  53,  -4, 113],
        [ -4,  55,   2,  98],
        [ -4,  58,   2,  98],
        [ -1,  59,   2,  89],
        [ -1,  59,  -1,  84],
        [ -1,  59,  -1,  84],
        [ -7,  63,  -1,  84],
        [ -7,  63,  -1,  84],
        [ -7,  63,   3,  83],
        [ -7,  63,   3,  83],
        [ -7,  63,   3,  83],
        [ -7,  63,   3,  83],
        [ -7,  63,   3,  83],
        [ -7,  63,   3,  83],
        [ -7,  63,   3,  83],
        [ -4,  70,   3,  83],
        [ -4,  70,   3,  83],
        [ -2,  73,   5,  90],
        [ -3,  79,  -4,  96],
        [-15,  98,  13, 135],
        [ -6,  93,  12, 128],
        [ -2,  89,  14, 118],
        [  5,  90,  10, 108],
        [  4,  86,   7, 105],
        [  4,  86,   7, 105],
        [  4,  86,  13, 100],
        [ -3,  82,  14,  96],
        [ -3,  82,  14,  96],
        [  6,  89,  16, 100],
        [  6,  89,  16, 100],
        [  7,  85,  17, 111]]), [17, 20, 12])
```

More feature sets

So far we have a simple feature set that is enough to get started modeling. However, we might get better results if we manipulate the raw values a bit more, so we will go ahead and set up some other options now for experimentation later. For example, we could normalize each speaker's range of motion with grouped statistics using Pandas stats (http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats) functions and pandas groupby (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html). Below is an example for finding the means of all speaker subgroups.

In [8]:
```
df_means = asl.df.groupby('speaker').mean()
df_means
```

Out[8]:

	left-x	left-y	right-x	right-y	nose-x	nose-y	grnd-ry	grnd-rx	grnd-ly	grnd-lx
speaker										
man-1	206.248203	218.679449	155.464350	150.371031	175.031756	61.642600	88.728430	-19.567406	157.036848	31.216447
woman-1	164.661438	161.271242	151.017865	117.332462	162.655120	57.245098	60.087364	-11.637255	104.026144	2.006318
woman-2	183.214509	176.527232	156.866295	119.835714	170.318973	58.022098	61.813616	-13.452679	118.505134	12.895536

To select a mean that matches by speaker, use the pandas map (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.map.html) method:

In [9]:
```
asl.df['left-x-mean']= asl.df['speaker'].map(df_means['left-x'])
asl.df.head()
```

Out[9]:

		left-x	left-y	right-x	right-y	nose-x	nose-y	speaker	grnd-ry	grnd-rx	grnd-ly	grnd-lx	left-x-mean
video	**frame**												
98	**0**	149	181	170	175	161	62	woman-1	113	9	119	-12	164.661438
	1	149	181	170	175	161	62	woman-1	113	9	119	-12	164.661438
	2	149	181	170	175	161	62	woman-1	113	9	119	-12	164.661438
	3	149	181	170	175	161	62	woman-1	113	9	119	-12	164.661438
	4	149	181	170	175	161	62	woman-1	113	9	119	-12	164.661438

Try it!

In [10]:
```
from asl_utils import test_std_tryit
# TODO Create a dataframe named `df_std` with standard deviations grouped by speaker
df_std = asl.df.groupby('speaker').std()
# test the code
test_std_tryit(df_std)
```

df_std

	left-x	left-y	right-x	right-y	nose-x	nose-y	grnd-ry	grnd-rx	grnd-ly	grnd-lx	left-x-mean
speaker											
man-1	15.154425	36.328485	18.901917	54.902340	6.654573	5.520045	53.487999	20.269032	36.572749	15.080360	0.0
woman-1	17.573442	26.594521	16.459943	34.667787	3.549392	3.538330	33.972660	16.764706	27.117393	17.328941	0.0
woman-2	15.388711	28.825025	14.890288	39.649111	4.099760	3.416167	39.128572	16.191324	29.320655	15.050938	0.0

Out[10]: Correct!

Features Implementation Submission

Implement four feature sets and answer the question that follows.

- normalized Cartesian coordinates
  - use *mean* and *standard deviation* statistics and the standard score (https://en.wikipedia.org/wiki/Standard_score) equation to account for speakers with different heights and arm length
- polar coordinates
  - calculate polar coordinates with Cartesian to polar equations (https://en.wikipedia.org/wiki/Polar_coordinate_system#Converting_between_polar_and_Cartesian_coordinates)
  - use the np.arctan2 (https://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.arctan2.html) function and *swap the x and y axes* to move the 0 to 2π discontinuity to 12 o'clock instead of 3 o'clock; in other words, the normal break in radians value from 0 to 2π occurs directly to the left of the speaker's nose, which may be in the signing area and interfere with results. By swapping the x and y axes, that discontinuity move to directly above the speaker's head, an area not generally used in signing.
- delta difference
  - as described in Thad's lecture, use the difference in values between one frame and the next frames as features
  - pandas diff method (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.diff.html) and fillna method (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html) will be helpful for this one
- custom features
  - These are your own design; combine techniques used above or come up with something else entirely. We look forward to seeing what you come up with! Some ideas to get you started:
    - normalize using a feature scaling equation (https://en.wikipedia.org/wiki/Feature_scaling)
    - normalize the polar coordinates
    - adding additional deltas

In [11]:
```
# TODO add features for normalized by speaker values of left, right, x, y
# Name these 'norm-rx', 'norm-ry', 'norm-lx', and 'norm-ly'
# using Z-score scaling (X-Xmean)/Xstd

# left-x-mean exists, add the rest
asl.df['left-y-mean'] = asl.df['speaker'].map(df_means['left-y'])
asl.df['right-x-mean'] = asl.df['speaker'].map(df_means['right-x'])
asl.df['right-y-mean'] = asl.df['speaker'].map(df_means['right-y'])

# Use the same way to create std for each tuple
asl.df['left-x-std'] = asl.df['speaker'].map(df_std['left-x'])
asl.df['left-y-std'] = asl.df['speaker'].map(df_std['left-y'])
asl.df['right-x-std'] = asl.df['speaker'].map(df_std['right-x'])
asl.df['right-y-std'] = asl.df['speaker'].map(df_std['right-y'])

features_norm = ['norm-rx', 'norm-ry', 'norm-lx','norm-ly']

# Create the norms
asl.df['norm-lx'] = (asl.df['left-x'] - asl.df['left-x-mean']) / asl.df['left-x-std']
asl.df['norm-ly'] = (asl.df['left-y'] - asl.df['left-y-mean']) / asl.df['left-y-std']
asl.df['norm-rx'] = (asl.df['right-x'] - asl.df['right-x-mean']) / asl.df['right-x-std']
asl.df['norm-ry'] = (asl.df['right-y'] - asl.df['right-y-mean']) / asl.df['right-y-std']
```

In [12]:
```
# TODO add features for polar coordinate values where the nose is the origin
# Name these 'polar-rr', 'polar-rtheta', 'polar-lr', and 'polar-ltheta'
# Note that 'polar-rr' and 'polar-rtheta' refer to the radius and angle

features_polar = ['polar-rr', 'polar-rtheta', 'polar-lr', 'polar-ltheta']

asl.df['polar-lr'] = np.sqrt(asl.df['grnd-lx'] ** 2 + asl.df['grnd-ly'] ** 2)
asl.df['polar-ltheta'] = np.arctan2(asl.df['grnd-lx'], asl.df['grnd-ly'])

asl.df['polar-rr'] = np.sqrt(asl.df['grnd-rx'] ** 2 + asl.df['grnd-ry'] ** 2)
asl.df['polar-rtheta'] = np.arctan2(asl.df['grnd-rx'], asl.df['grnd-ry'])
```

In [13]:
```
# TODO add features for left, right, x, y differences by one time step, i.e. the "delta" values discussed in
 the lecture
# Name these 'delta-rx', 'delta-ry', 'delta-lx', and 'delta-ly'

features_delta = ['delta-rx', 'delta-ry', 'delta-lx', 'delta-ly']

# Create a set structure to get unique video indexes
video_set = set(asl.df.index.get_level_values('video'))
for v in video_set:
    # for each video
    asl.df.loc[v,'delta-rx'] = np.array(asl.df.loc[v]['right-x'].diff())
    asl.df.loc[v,'delta-ry'] = np.array(asl.df.loc[v]['right-y'].diff())
    asl.df.loc[v,'delta-lx'] = np.array(asl.df.loc[v]['left-x'].diff())
    asl.df.loc[v,'delta-ly'] = np.array(asl.df.loc[v]['left-y'].diff())

asl.df[features_delta] = asl.df[features_delta].fillna(0).astype(int)
```

In [14]:
```
# TODO add features of your own design, which may be a combination of the above or something else
# Name these whatever you would like

# The features created so far cover the initial and end positions of right and left hands, as
# well as the movement. I think that's enough and what we can play with is the combination of
#the features.

# TODO define a list named 'features_custom' for building the training set
features_custom = ['grnd-rx', 'grnd-ry', 'grnd-lx', 'grnd-ly', 'delta-rx', 'delta-ry', 'delta-lx', 'delta-
ly']
```

Question 1: What custom features did you choose for the features_custom set and why?

Answer 1: The ground and delta features. From my point of view, the start, end positions of hands, how the hands move and how fast the hands move are important to identify the signs.

Features Unit Testing

Run the following unit tests as a sanity check on the defined "ground", "norm", "polar", and 'delta" feature sets. The test simply looks for some valid values but is not exhaustive. However, the project should not be submitted if these tests don't pass.

In [15]:
```
import unittest
# import numpy as np

class TestFeatures(unittest.TestCase):

    def test_features_ground(self):
        sample = (asl.df.ix[98, 1][features_ground]).tolist()
        self.assertEqual(sample, [9, 113, -12, 119])

    def test_features_norm(self):
        sample = (asl.df.ix[98, 1][features_norm]).tolist()
        np.testing.assert_almost_equal(sample, [ 1.153,  1.663, -0.891,  0.742], 3)

    def test_features_polar(self):
        sample = (asl.df.ix[98,1][features_polar]).tolist()
        np.testing.assert_almost_equal(sample, [113.3578, 0.0794, 119.603, -0.1005], 3)

    def test_features_delta(self):
        sample = (asl.df.ix[98, 0][features_delta]).tolist()
        self.assertEqual(sample, [0, 0, 0, 0])
        sample = (asl.df.ix[98, 18][features_delta]).tolist()
        self.assertTrue(sample in [[-16, -5, -2, 4], [-14, -9, 0, 0]], "Sample value found was {}".format(sam
ple))

suite = unittest.TestLoader().loadTestsFromModule(TestFeatures())
unittest.TextTestRunner().run(suite)
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.014s

OK
```

Out[15]: <unittest.runner.TextTestResult run=4 errors=0 failures=0>

PART 2: Model Selection

Model Selection Tutorial

The objective of Model Selection is to tune the number of states for each word HMM prior to testing on unseen data. In this section you will explore three methods:

- Log likelihood using cross-validation folds (CV)
- Bayesian Information Criterion (BIC)
- Discriminative Information Criterion (DIC)

Train a single word

Now that we have built a training set with sequence data, we can "train" models for each word. As a simple starting example, we train a single word using Gaussian hidden Markov models (HMM). By using the `fit` method during training, the Baum-Welch Expectation-Maximization (https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm) (EM) algorithm is invoked iteratively to find the best estimate for the model *for the number of hidden states specified* from a group of sample seequences. For this example, we *assume* the correct number of hidden states is 3, but that is just a guess. How do we know what the "best" number of states for training is? We will need to find some model selection technique to choose the best parameter.

In [16]:
```
import warnings
from hmmlearn.hmm import GaussianHMM

def train_a_word(word, num_hidden_states, features):

    warnings.filterwarnings("ignore", category=DeprecationWarning)
    training = asl.build_training(features)
    X, lengths = training.get_word_Xlengths(word)
    model = GaussianHMM(n_components=num_hidden_states, n_iter=1000).fit(X, lengths)
    logL = model.score(X, lengths)
    return model, logL

demoword = 'BOOK'
model, logL = train_a_word(demoword, 3, features_ground)
print("Number of states trained in model for {} is {}".format(demoword, model.n_components))
print("logL = {}".format(logL))
```

```
Number of states trained in model for BOOK is 3
logL = -2331.1138127433205
```

The HMM model has been trained and information can be pulled from the model, including means and variances for each feature and hidden state. The log likelihood (http://math.stackexchange.com/questions/892832/why-we-consider-log-likelihood-instead-of-likelihood-in-gaussian-distribution) for any individual sample or group of samples can also be calculated with the `score` method.

In [17]:
```
def show_model_stats(word, model):
    print("Number of states trained in model for {} is {}".format(word, model.n_components))
    variance=np.array([np.diag(model.covars_[i]) for i in range(model.n_components)])
    for i in range(model.n_components):  # for each hidden state
        print("hidden state #{}".format(i))
        print("mean = ", model.means_[i])
        print("variance = ", variance[i])
        print()

show_model_stats(demoword, model)
```

```
Number of states trained in model for BOOK is 3
hidden state #0
mean =  [ -3.46504869  50.66686933  14.02391587  52.04731066]
variance =  [ 49.12346305  43.04799144  39.35109609  47.24195772]

hidden state #1
mean =  [ -11.45300909   94.109178     19.03512475  102.2030162 ]
variance =  [  77.403668    203.35441965   26.68898447  156.12444034]

hidden state #2
mean =  [ -1.12415027  69.44164191  17.02866283  77.7231196 ]
variance =  [ 19.70434594  16.83041492  30.51552305  11.03678246]
```

Try it!

Experiment by changing the feature set, word, and/or num_hidden_states values in the next cell to see changes in values.

In [18]:
```
my_testword = 'CHOCOLATE'
model, logL = train_a_word(my_testword, 3, features_custom) # Experiment here with different parameters
show_model_stats(my_testword, model)
print("logL = {}".format(logL))
```

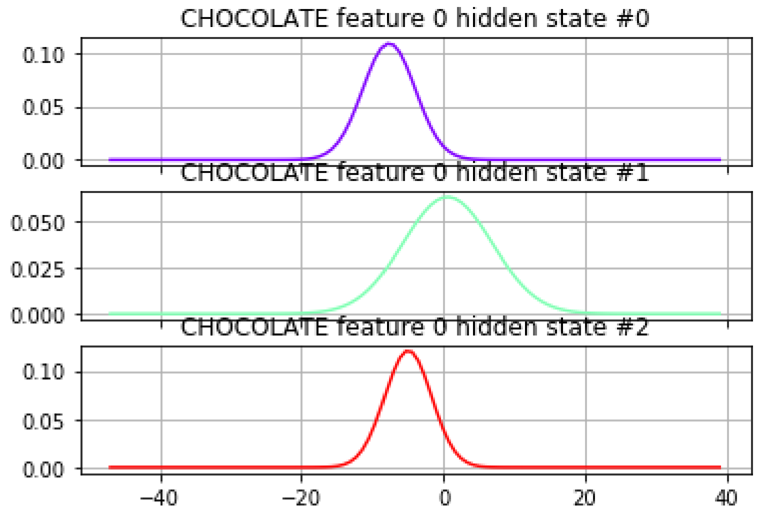
```
Number of states trained in model for CHOCOLATE is 3
hidden state #0
mean =  [ -7.66377175  61.24677787   4.1192691   78.66546514  -0.61190197
   1.67253245   0.16138129   0.97598705]
variance =  [ 13.52380134  64.32641516  11.74963606  59.964627     7.33595022
  10.46092249   4.95652426   4.85607291]

hidden state #1
mean =  [  5.83333333e-01   8.79166667e+01   1.27500000e+01   1.08500000e+02
   2.25000000e+00  -1.16666667e+00  -8.33333333e-02  -2.83333333e+00]
variance =  [  39.41055556   18.74388889    9.855       144.4175       22.355
   10.97305556    7.24388889   43.30638889]

hidden state #2
mean =  [ -4.97605651e+00   5.30673351e+01   3.52558074e+00   9.55141867e+01
   8.80966899e-01   1.44158284e+00   3.51936073e-03  -7.44217698e+00]
variance =  [  10.85612477   17.0524101    21.29006447  231.62097862    2.18795279
    2.56774025    6.07695553   26.83459275]

logL = -1096.9061507105864
```

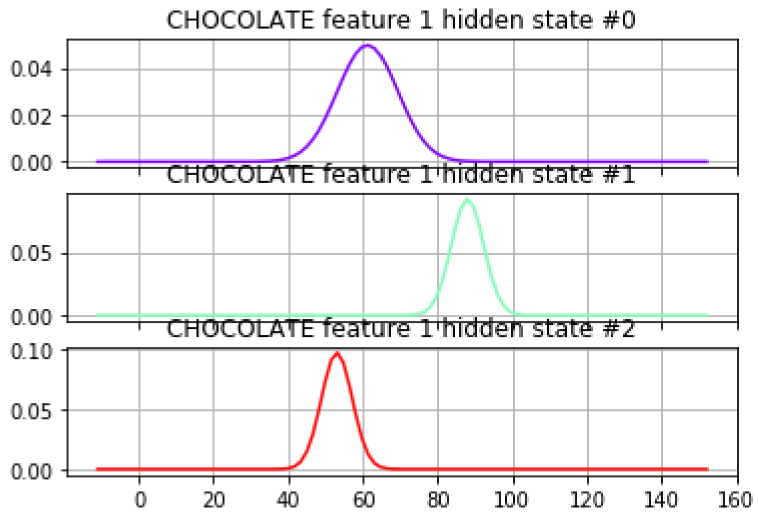
Visualize the hidden states

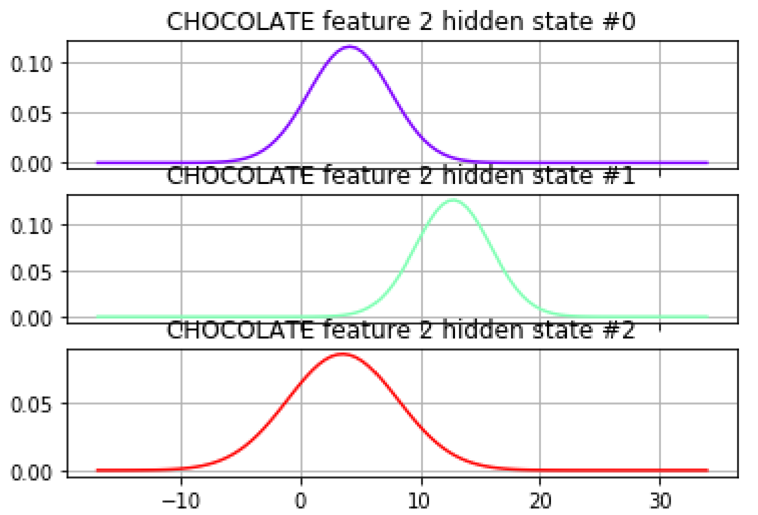
We can plot the means and variances for each state and feature. Try varying the number of states trained for the HMM model and examine the variances. Are there some models that are "better" than others? How can you tell? We would like to hear what you think in the classroom online.

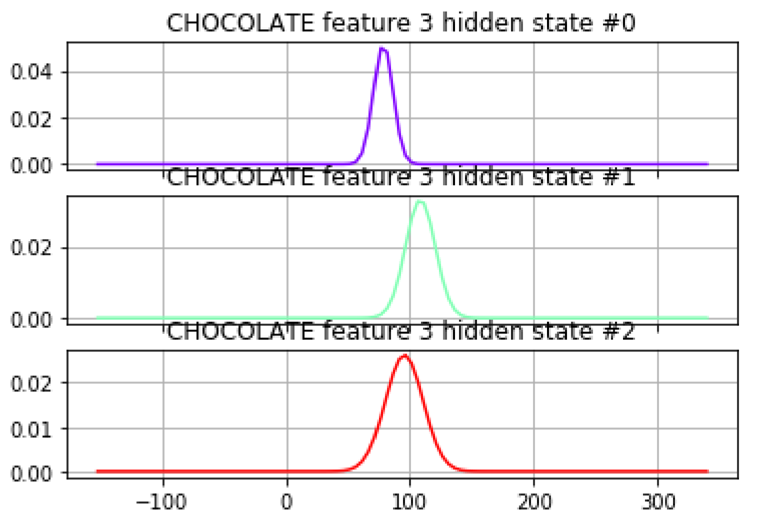
In [19]:
```
%matplotlib inline
```

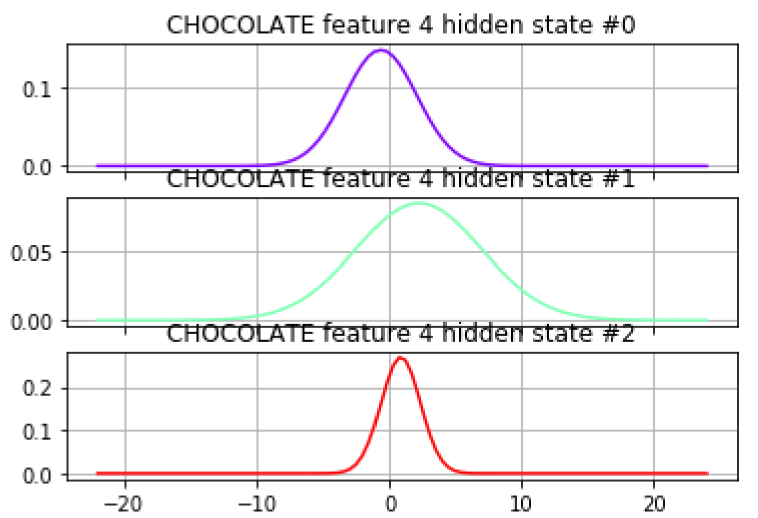
In [20]:
```
import math
from matplotlib import (cm, pyplot as plt, mlab)

def visualize(word, model):
    """ visualize the input model for a particular word """
    variance=np.array([np.diag(model.covars_[i]) for i in range(model.n_components)])
    figures = []
    for parm_idx in range(len(model.means_[0])):
        xmin = int(min(model.means_[:,parm_idx]) - max(variance[:,parm_idx]))
        xmax = int(max(model.means_[:,parm_idx]) + max(variance[:,parm_idx]))
        fig, axs = plt.subplots(model.n_components, sharex=True, sharey=False)
        colours = cm.rainbow(np.linspace(0, 1, model.n_components))
        for i, (ax, colour) in enumerate(zip(axs, colours)):
            x = np.linspace(xmin, xmax, 100)
            mu = model.means_[i,parm_idx]
            sigma = math.sqrt(np.diag(model.covars_[i])[parm_idx])
            ax.plot(x, mlab.normpdf(x, mu, sigma), c=colour)
            ax.set_title("{} feature {} hidden state #{}".format(word, parm_idx, i))

            ax.grid(True)
        figures.append(plt)
    for p in figures:
        p.show()

visualize(my_testword, model)
```

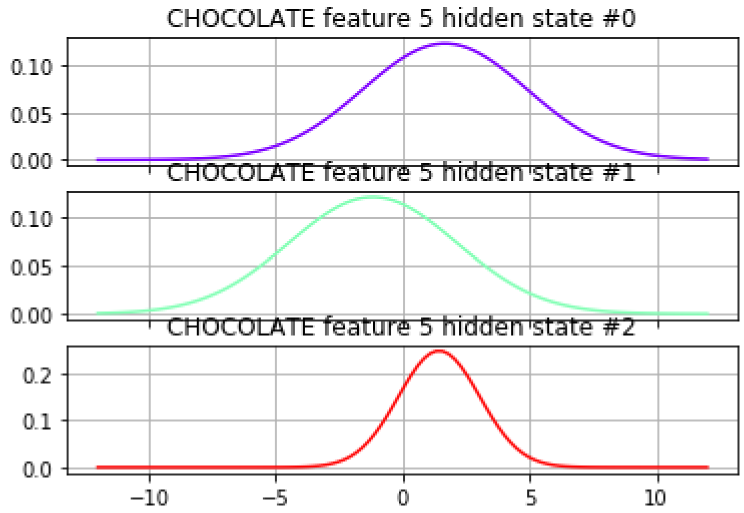

CHOCOLATE feature 0 hidden state #0
CHOCOLATE feature 0 hidden state #1
CHOCOLATE feature 0 hidden state #2

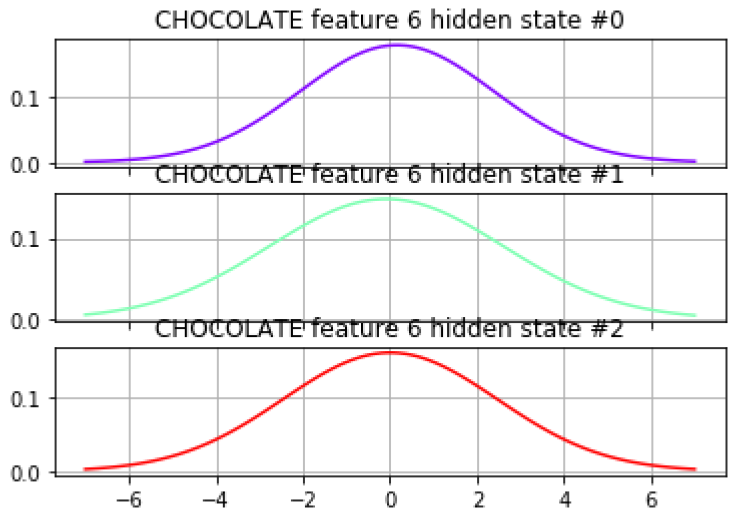

CHOCOLATE feature 1 hidden state #0
CHOCOLATE feature 1 hidden state #1
CHOCOLATE feature 1 hidden state #2

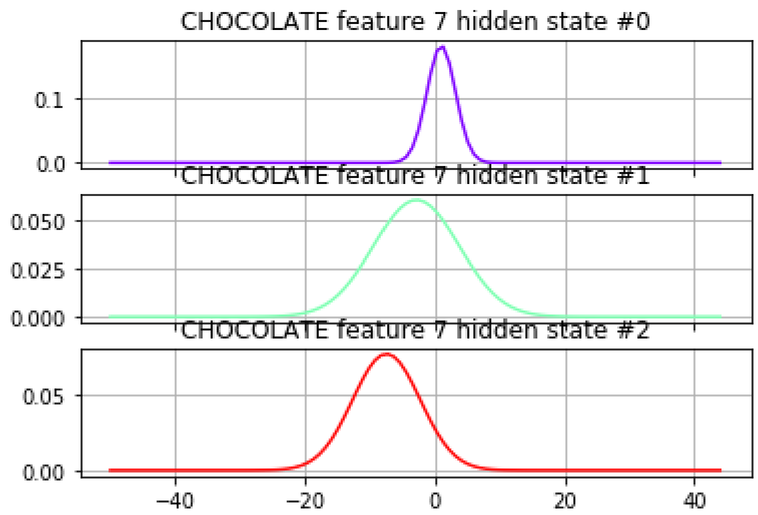

CHOCOLATE feature 2 hidden state #0
CHOCOLATE feature 2 hidden state #1
CHOCOLATE feature 2 hidden state #2


CHOCOLATE feature 3 hidden state #0
CHOCOLATE feature 3 hidden state #1
CHOCOLATE feature 3 hidden state #2


CHOCOLATE feature 4 hidden state #0
CHOCOLATE feature 4 hidden state #1
CHOCOLATE feature 4 hidden state #2







ModelSelector class

Review the `SelectorModel` class from the codebase found in the `my_model_selectors.py` module. It is designed to be a strategy pattern for choosing different model selectors. For the project submission in this section, subclass `SelectorModel` to implement the following model selectors. In other words, you will write your own classes/functions in the `my_model_selectors.py` module and run them from this notebook:

- `SelectorCV`: Log likelihood with CV
- `SelectorBIC`: BIC
- `SelectorDIC`: DIC

You will train each word in the training set with a range of values for the number of hidden states, and then score these alternatives with the model selector, choosing the "best" according to each strategy. The simple case of training with a constant value for `n_components` can be called using the provided `SelectorConstant` subclass as follow:

In [21]:
```
from my_model_selectors import SelectorConstant

training = asl.build_training(features_ground)  # Experiment here with different feature sets defined in part
 1
word = 'VEGETABLE' # Experiment here with different words
model = SelectorConstant(training.get_all_sequences(), training.get_all_Xlengths(), word, n_constant=3).selec
t()
print("Number of states trained in model for {} is {}".format(word, model.n_components))
```

```
INFO:root:Started my_model_selectors.py
INFO:root:Finished my_model_selectors.py
```

```
Number of states trained in model for VEGETABLE is 3
```

Cross-validation folds

If we simply score the model with the Log Likelihood calculated from the feature sequences it has been trained on, we should expect that more complex models will have higher likelihoods. However, that doesn't tell us which would have a better likelihood score on unseen data. The model will likely be overfit as complexity is added. To estimate which topology model is better using only the training data, we can compare scores using cross-validation. One technique for cross-validation is to break the training set into "folds" and rotate which fold is left out of training. The "left out" fold scored. This gives us a proxy method of finding the best model to use on "unseen data". In the following example, a set of word sequences is broken into three folds using the [scikit-learn Kfold (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html)](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html) class object. When you implement `SelectorCV`, you will use this technique.

In [22]:
```
from sklearn.model_selection import KFold

training = asl.build_training(features_ground) # Experiment here with different feature sets
word = 'VEGETABLE' # Experiment here with different words
word_sequences = training.get_word_sequences(word)
split_method = KFold()
for cv_train_idx, cv_test_idx in split_method.split(word_sequences):
    print("Train fold indices:{} Test fold indices:{}".format(cv_train_idx, cv_test_idx))  # view indices of
 the folds
```

```
Train fold indices:[2 3 4 5] Test fold indices:[0 1]
Train fold indices:[0 1 4 5] Test fold indices:[2 3]
Train fold indices:[0 1 2 3] Test fold indices:[4 5]
```

Tip: In order to run `hmmlearn` training using the X,lengths tuples on the new folds, subsets must be combined based on the indices given for the folds. A helper utility has been provided in the `asl_utils` module named `combine_sequences` for this purpose.

Scoring models with other criterion

Scoring model topologies with **BIC** balances fit and complexity within the training set for each word. In the BIC equation, a penalty term penalizes complexity to avoid overfitting, so that it is not necessary to also use cross-validation in the selection process. There are a number of references on the internet for this criterion. These [slides (http://www2.imm.dtu.dk/courses/02433/doc/ch6_slides.pdf)](http://www2.imm.dtu.dk/courses/02433/doc/ch6_slides.pdf) include a formula you may find helpful for your implementation.

The advantages of scoring model topologies with **DIC** over BIC are presented by Alain Biem in this [reference (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.6208&rep=rep1&type=pdf)](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.6208&rep=rep1&type=pdf) (also found [here (https://pdfs.semanticscholar.org/ed3d/7c4a5f607201f3848d4c02dd9ba17c791fc2.pdf)](https://pdfs.semanticscholar.org/ed3d/7c4a5f607201f3848d4c02dd9ba17c791fc2.pdf)). DIC scores the discriminant ability of a training set for one word against competing words. Instead of a penalty term for complexity, it provides a penalty if model liklihoods for non-matching words are too similar to model likelihoods for the correct word in the word set.

Model Selection Implementation Submission

Implement `SelectorCV`, `SelectorBIC`, and `SelectorDIC` classes in the `my_model_selectors.py` module. Run the selectors on the following five words. Then answer the questions about your results.

Tip: The `hmmlearn` library may not be able to train or score all models. Implement try/except contructs as necessary to eliminate non-viable models from consideration.

In [23]:
```
words_to_train = ['FISH', 'BOOK', 'VEGETABLE', 'FUTURE', 'JOHN']
import timeit
```

In [24]:
```
# TODO: Implement SelectorCV in my_model_selector.py
from my_model_selectors import SelectorCV

training = asl.build_training(features_custom)  # Experiment here with different feature sets defined in part
 1
sequences = training.get_all_sequences()
Xlengths = training.get_all_Xlengths()
for word in words_to_train:
    start = timeit.default_timer()
    model = SelectorCV(sequences, Xlengths, word,
                    min_n_components=2, max_n_components=15, random_state = 14).select()
    end = timeit.default_timer()-start
    if model is not None:
        print("Training complete for {} with {} states with time {} seconds".format(word, model.n_components,
 end))
    else:
        print("Training failed for {}".format(word))
```

```
Training complete for FISH with 13 states with time 0.3578397660749033 seconds
Training complete for BOOK with 4 states with time 4.490520740044303 seconds
Training complete for VEGETABLE with 2 states with time 1.9493374839657918 seconds
Training complete for FUTURE with 2 states with time 4.083184369024821 seconds
Training complete for JOHN with 15 states with time 45.669709142064676 seconds
```

In [25]:
```
# TODO: Implement SelectorBIC in module my_model_selectors.py
from my_model_selectors import SelectorBIC

training = asl.build_training(features_custom)  # Experiment here with different feature sets defined in part
 1
sequences = training.get_all_sequences()
Xlengths = training.get_all_Xlengths()
for word in words_to_train:
    start = timeit.default_timer()
    model = SelectorBIC(sequences, Xlengths, word,
                        min_n_components=2, max_n_components=15, random_state = 14).select()
    end = timeit.default_timer()-start
    if model is not None:
        print("Training complete for {} with {} states with time {} seconds".format(word, model.n_components,
 end))
    else:
        print("Training failed for {}".format(word))
```

```
Training complete for FISH with 4 states with time 0.35951623308937997 seconds
Training complete for BOOK with 7 states with time 3.277872535982169 seconds
Training complete for VEGETABLE with 7 states with time 0.8460396600421518 seconds
Training complete for FUTURE with 2 states with time 2.051487760967575 seconds
Training complete for JOHN with 14 states with time 21.50414330197964 seconds
```

In [26]:
```
# TODO: Implement SelectorDIC in module my_model_selectors.py
from my_model_selectors import SelectorDIC

training = asl.build_training(features_custom)  # Experiment here with different feature sets defined in part
 1
sequences = training.get_all_sequences()
Xlengths = training.get_all_Xlengths()
for word in words_to_train:
    start = timeit.default_timer()
    model = SelectorDIC(sequences, Xlengths, word,
                    min_n_components=2, max_n_components=15, random_state = 14).select()
    end = timeit.default_timer()-start
    if model is not None:
        print("Training complete for {} with {} states with time {} seconds".format(word, model.n_components,
 end))
    else:
        print("Training failed for {}".format(word))
```

```
Training complete for FISH with 4 states with time 0.693581396015361 seconds
Training complete for BOOK with 15 states with time 5.89467791991774 seconds
Training complete for VEGETABLE with 10 states with time 2.1892479510279372 seconds
Training complete for FUTURE with 15 states with time 4.549445662996732 seconds
Training complete for JOHN with 15 states with time 24.595467517036013 seconds
```

Question 2: Compare and contrast the possible advantages and disadvantages of the various model selectors implemented.

Answer 2: Based on the score formulas, I assume the time spent for each selector should be CV > DIC > BIC. And the testing scenarios do follow this observation. The BIC formula is simple and easy to implement. The complexity of the model affects the score. DIC considers not only the model itself, it also uses all the other available words to "test" the model during the selection. A good model should resturn a high likelihood when fit in the trained data, it should also return a low likelihood when fit the data other than the trained data. Thus, in this project, I think DIC will return the best results. CV may work good for a large scale of trained data, but in the project I've meet many models are only trained with 2 samples.

Model Selector Unit Testing

Run the following unit tests as a sanity check on the implemented model selectors. The test simply looks for valid interfaces but is not exhaustive. However, the project should not be submitted if these tests don't pass.

In [27]:
```
from asl_test_model_selectors import TestSelectors
suite = unittest.TestLoader().loadTestsFromModule(TestSelectors())
unittest.TextTestRunner().run(suite)
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 48.793s

OK
```

Out[27]:
```
<unittest.runner.TextTestResult run=4 errors=0 failures=0>
```

PART 3: Recognizer

The objective of this section is to "put it all together". Using the four feature sets created and the three model selectors, you will experiment with the models and present your results. Instead of training only five specific words as in the previous section, train the entire set with a feature set and model selector strategy.

Recognizer Tutorial

Train the full training set

The following example trains the entire set with the example `features_ground` and `SelectorConstant` features and model selector. Use this pattern for you experimentation and final submission cells.

In [28]:
```
# autoreload for automatically reloading changes made in my_model_selectors and my_recognizer
%load_ext autoreload
%autoreload 2

from my_model_selectors import SelectorConstant

def train_all_words(features, model_selector):
    training = asl.build_training(features)  # Experiment here with different feature sets defined in part 1
    sequences = training.get_all_sequences()
    Xlengths = training.get_all_Xlengths()
    model_dict = {}
    for word in training.words:
        model = model_selector(sequences, Xlengths, word,
                        n_constant=3).select()
        model_dict[word]=model
    return model_dict

models = train_all_words(features_ground, SelectorConstant)
print("Number of word models returned = {}".format(len(models)))
```

```
Number of word models returned = 112
```

Load the test set

The `build_test` method in `ASLdb` is similar to the `build_training` method already presented, but there are a few differences:

- the object is type `SinglesData`
- the internal dictionary keys are the index of the test word rather than the word itself
- the getter methods are `get_all_sequences`, `get_all_Xlengths`, `get_item_sequences` and `get_item_Xlengths`

In [29]:
```
test_set = asl.build_test(features_ground)
print("Number of test set items: {}".format(test_set.num_items))
print("Number of test set sentences: {}".format(len(test_set.sentences_index)))
```

```
Number of test set items: 178
Number of test set sentences: 40
```

Recognizer Implementation Submission

For the final project submission, students must implement a recognizer following guidance in the `my_recognizer.py` module. Experiment with the four feature sets and the three model selection methods (that's 12 possible combinations). You can add and remove cells for experimentation or run the recognizers locally in some other way during your experiments, but retain the results for your discussion. For submission, you will provide code cells of **only three** interesting combinations for your discussion (see questions below). At least one of these should produce a word error rate of less than 60%, i.e. WER < 0.60 .

Tip: The hmmlearn library may not be able to train or score all models. Implement try/except contructs as necessary to eliminate non-viable models from consideration.

In [30]:
```
# TODO implement the recognize method in my_recognizer
from my_recognizer import recognize
from asl_utils import show_errors
```

```
INFO:root:Started my_recognizer.py
INFO:root:Finished my_recognizer.py
```

In [31]:
```
# TODO Choose a feature set and model selector
features = features_ground # change as needed
model_selector = SelectorBIC # change as needed

# TODO Recognize the test set and display the result with the show_errors method
models = train_all_words(features, model_selector)
test_set = asl.build_test(features)
probabilities, guesses = recognize(models, test_set)
show_errors(guesses, test_set)
```

```
**** WER = 0.550561797752809
Total correct: 80 out of 178
Video  Recognized                                                    Correct
=====================================================================================================
    2: JOHN WRITE *NEW                                               JOHN WRITE HOMEWORK
    7: *SOMETHING-ONE *GO1 GO *ARRIVE                                JOHN CAN GO CAN
   12: *IX *WHAT *CAN CAN                                            JOHN CAN GO CAN
   21: JOHN *WRITE *JOHN *FUTURE *CAR *TEACHER *VISIT *WHO           JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN *IX IX *LIKE IX                                          JOHN LIKE IX IX IX
   28: JOHN *WHO IX *LIKE *LOVE                                      JOHN LIKE IX IX IX
   30: JOHN LIKE *MARY *MARY *MARY                                   JOHN LIKE IX IX IX
   36: *VISIT *VISIT *IX *GIVE *MARY *IX                             MARY VEGETABLE KNOW IX LIKE CORN1
   40: *MARY *GO *GIVE MARY *MARY                                    JOHN IX THINK MARY LOVE
   43: JOHN *IX BUY HOUSE                                            JOHN MUST BUY HOUSE
   50: *JOHN *SEE BUY CAR *NEW                                       FUTURE JOHN BUY CAR SHOULD
   54: JOHN SHOULD NOT BUY HOUSE                                     JOHN SHOULD NOT BUY HOUSE
   57: *MARY *VISIT VISIT MARY                                       JOHN DECIDE VISIT MARY
   67: *SHOULD *JOHN *WHO BUY HOUSE                                  JOHN FUTURE NOT BUY HOUSE
   71: JOHN *FUTURE VISIT MARY                                       JOHN WILL VISIT MARY
   74: *IX *VISIT VISIT MARY                                         JOHN NOT VISIT MARY
   77: *JOHN BLAME *LOVE                                             ANN BLAME MARY
   84: *JOHN *ARRIVE *GIVE1 BOOK                                     IX-1P FIND SOMETHING-ONE BOOK
   89: *MARY *POSS *IX *IX IX *ARRIVE *BOOK                          JOHN IX GIVE MAN IX NEW COAT
   90: JOHN *SOMETHING-ONE IX *IX *VISIT *ARRIVE                     JOHN GIVE IX SOMETHING-ONE WOMAN BOOK
   92: JOHN *SHOULD IX *IX *IX BOOK                                  JOHN GIVE IX SOMETHING-ONE WOMAN BOOK
  100: *IX NEW CAR BREAK-DOWN                                        POSS NEW CAR BREAK-DOWN
  105: JOHN *FRANK                                                   JOHN LEG
  107: JOHN *GO *ARRIVE HAVE *JOHN                                   JOHN POSS FRIEND HAVE CANDY
  108: *WHO *LOVE                                                    WOMAN ARRIVE
  113: IX CAR *CAR *MARY *BOX                                        IX CAR BLUE SUE BUY
  119: *VISIT *BUY1 IX *BOX *GO                                      SUE BUY IX CAR BLUE
  122: JOHN *GIVE1 BOOK                                              JOHN READ BOOK
  139: JOHN *BUY1 WHAT *GIVE1 BOOK                                   JOHN BUY WHAT YESTERDAY BOOK
  142: JOHN *STUDENT YESTERDAY WHAT BOOK                             JOHN BUY YESTERDAY WHAT BOOK
  158: LOVE JOHN WHO                                                 LOVE JOHN WHO
  167: JOHN *MARY *VISIT LOVE MARY                                   JOHN IX SAY LOVE MARY
  171: JOHN MARY BLAME                                               JOHN MARY BLAME
  174: *CAN *GIVE1 GIVE1 *YESTERDAY *WHAT                            PEOPLE GROUP GIVE1 JANA TOY
  181: JOHN *BOX                                                     JOHN ARRIVE
  184: *GIVE BOY *GIVE1 TEACHER APPLE                                ALL BOY GIVE TEACHER APPLE
  189: JOHN *SOMETHING-ONE *VISIT BOX                                JOHN GIVE GIRL BOX
  193: JOHN *SOMETHING-ONE *VISIT BOX                                JOHN GIVE GIRL BOX
  199: *JOHN CHOCOLATE *GO                                           LIKE CHOCOLATE WHO
  201: JOHN *MARY *LOVE *JOHN BUY HOUSE                              JOHN TELL MARY IX-1P BUY HOUSE
```

In [32]:
```
# TODO Choose a feature set and model selector
features = features_polar # change as needed
model_selector = SelectorDIC # change as needed

# TODO Recognize the test set and display the result with the show_errors method
models = train_all_words(features, model_selector)
test_set = asl.build_test(features)
probabilities, guesses = recognize(models, test_set)
show_errors(guesses, test_set)
```

```
**** WER = 0.5449438202247191
Total correct: 81 out of 178
Video  Recognized                                                    Correct
=====================================================================================================
    2: JOHN *NEW *GIVE1                                              JOHN WRITE HOMEWORK
    7: JOHN CAN GO CAN                                               JOHN CAN GO CAN
   12: JOHN *WHAT *JOHN CAN                                          JOHN CAN GO CAN
   21: JOHN *NEW *JOHN *PREFER *CAR *WHAT *FUTURE *WHO               JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN *IX IX *WHO IX                                           JOHN LIKE IX IX IX
   28: JOHN *FUTURE IX *FUTURE *LOVE                                 JOHN LIKE IX IX IX
   30: JOHN LIKE *MARY *MARY *MARY                                   JOHN LIKE IX IX IX
   36: *IX *VISIT *GIVE *GIVE *MARY *MARY                            MARY VEGETABLE KNOW IX LIKE CORN1
   40: JOHN *GO *GIVE *JOHN *MARY                                    JOHN IX THINK MARY LOVE
   43: JOHN *IX BUY HOUSE                                            JOHN MUST BUY HOUSE
   50: *JOHN *SEE BUY CAR *JOHN                                      FUTURE JOHN BUY CAR SHOULD
   54: JOHN SHOULD NOT BUY HOUSE                                     JOHN SHOULD NOT BUY HOUSE
   57: *MARY *GO *GO MARY                                            JOHN DECIDE VISIT MARY
   67: JOHN FUTURE *MARY BUY HOUSE                                   JOHN FUTURE NOT BUY HOUSE
   71: JOHN *FUTURE *GIVE1 MARY                                      JOHN WILL VISIT MARY
   74: *IX *GO *GO *VISIT                                            JOHN NOT VISIT MARY
   77: *JOHN *GIVE1 MARY                                             ANN BLAME MARY
   84: *HOMEWORK *GIVE1 *POSS *COAT                                  IX-1P FIND SOMETHING-ONE BOOK
   89: *GIVE *GIVE *WOMAN *IX IX *ARRIVE *BOOK                       JOHN IX GIVE MAN IX NEW COAT
   90: JOHN GIVE IX SOMETHING-ONE WOMAN *ARRIVE                      JOHN GIVE IX SOMETHING-ONE WOMAN BOOK
   92: JOHN *WOMAN IX *IX *IX BOOK                                   JOHN GIVE IX SOMETHING-ONE WOMAN BOOK
  100: POSS NEW CAR BREAK-DOWN                                       POSS NEW CAR BREAK-DOWN
  105: JOHN *SEE                                                     JOHN LEG
  107: JOHN POSS *HAVE HAVE *MARY                                    JOHN POSS FRIEND HAVE CANDY
  108: *LOVE *LOVE                                                   WOMAN ARRIVE
  113: IX CAR *IX *MARY *JOHN                                        IX CAR BLUE SUE BUY
  119: *MARY *BUY1 IX *BLAME *IX                                     SUE BUY IX CAR BLUE
  122: JOHN *GIVE1 BOOK                                              JOHN READ BOOK
  139: JOHN *ARRIVE WHAT *MARY *ARRIVE                               JOHN BUY WHAT YESTERDAY BOOK
  142: JOHN BUY YESTERDAY WHAT BOOK                                  JOHN BUY YESTERDAY WHAT BOOK
  158: LOVE JOHN WHO                                                 LOVE JOHN WHO
  167: JOHN *MARY *VISIT LOVE MARY                                   JOHN IX SAY LOVE MARY
  171: *IX MARY BLAME                                                JOHN MARY BLAME
  174: *JOHN *JOHN GIVE1 *YESTERDAY *JOHN                            PEOPLE GROUP GIVE1 JANA TOY
  181: *EAT ARRIVE                                                   JOHN ARRIVE
  184: *GO BOY *GIVE1 TEACHER *YESTERDAY                             ALL BOY GIVE TEACHER APPLE
  189: *MARY *GO *YESTERDAY BOX                                      JOHN GIVE GIRL BOX
  193: JOHN *GO *YESTERDAY BOX                                       JOHN GIVE GIRL BOX
  199: *JOHN *STUDENT *GO                                            LIKE CHOCOLATE WHO
  201: JOHN *GIVE *WOMAN *JOHN BUY HOUSE                             JOHN TELL MARY IX-1P BUY HOUSE
```

In [33]:
```
# TODO Choose a feature set and model selector
features = features_custom # change as needed
model_selector = SelectorConstant # change as needed

# TODO Recognize the test set and display the result with the show_errors method
models = train_all_words(features, model_selector)
test_set = asl.build_test(features)
probabilities, guesses = recognize(models, test_set)
show_errors(guesses, test_set)
```

```
**** WER = 0.48314606741573035
Total correct: 92 out of 178
Video  Recognized                                                    Correct
=====================================================================================================
    2: JOHN WRITE HOMEWORK                                           JOHN WRITE HOMEWORK
    7: JOHN *HAVE GO *ARRIVE                                         JOHN CAN GO CAN
   12: JOHN CAN *WHAT CAN                                            JOHN CAN GO CAN
   21: JOHN *VIDEOTAPE WONT *WHO *CAR *CAR *FUTURE *MARY             JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN LIKE *MARY *TELL *MARY                                   JOHN LIKE IX IX IX
   28: JOHN *WHO *MARY *LIKE *MARY                                   JOHN LIKE IX IX IX
   30: JOHN LIKE *MARY *MARY IX                                      JOHN LIKE IX IX IX
   36: MARY *MARY *JOHN *GIVE *MARY *MARY                            MARY VEGETABLE KNOW IX LIKE CORN1
   40: JOHN *GIVE *CORN MARY *MARY                                   JOHN IX THINK MARY LOVE
   43: JOHN *SHOULD BUY HOUSE                                        JOHN MUST BUY HOUSE
   50: *JOHN *POSS BUY CAR SHOULD                                    FUTURE JOHN BUY CAR SHOULD
   54: JOHN *JOHN *WHO BUY HOUSE                                     JOHN SHOULD NOT BUY HOUSE
   57: *MARY *MARY *IX MARY                                          JOHN DECIDE VISIT MARY
   67: JOHN *JOHN *MARY BUY HOUSE                                    JOHN FUTURE NOT BUY HOUSE
   71: JOHN *JOHN VISIT MARY                                         JOHN WILL VISIT MARY
   74: JOHN *MARY *MARY MARY                                         JOHN NOT VISIT MARY
   77: *JOHN BLAME MARY                                              ANN BLAME MARY
   84: *JOHN *ARRIVE *GO BOOK                                        IX-1P FIND SOMETHING-ONE BOOK
   89: *WHO IX *IX *IX IX NEW COAT                                   JOHN IX GIVE MAN IX NEW COAT
   90: JOHN *IX IX *IX *IX *COAT                                     JOHN GIVE IX SOMETHING-ONE WOMAN BOOK
   92: JOHN GIVE IX *WOMAN WOMAN BOOK                                JOHN GIVE IX SOMETHING-ONE WOMAN BOOK
  100: POSS NEW CAR BREAK-DOWN                                       POSS NEW CAR BREAK-DOWN
  105: JOHN *VEGETABLE                                               JOHN LEG
  107: JOHN *IX *HAVE *IX *JOHN                                      JOHN POSS FRIEND HAVE CANDY
  108: *WHO *BOOK                                                    WOMAN ARRIVE
  113: IX CAR *IX *JOHN *IX                                          IX CAR BLUE SUE BUY
  119: *MARY *BUY1 IX CAR *IX                                        SUE BUY IX CAR BLUE
  122: JOHN *GIVE1 BOOK                                              JOHN READ BOOK
  139: JOHN *BUY1 *CAR *MARY BOOK                                    JOHN BUY WHAT YESTERDAY BOOK
  142: JOHN BUY YESTERDAY WHAT BOOK                                  JOHN BUY YESTERDAY WHAT BOOK
  158: LOVE JOHN WHO                                                 LOVE JOHN WHO
  167: JOHN IX *MARY LOVE MARY                                       JOHN IX SAY LOVE MARY
  171: *MARY *IX BLAME                                               JOHN MARY BLAME
  174: *GIVE1 *GIVE3 GIVE1 *MARY *BLAME                              PEOPLE GROUP GIVE1 JANA TOY
  181: JOHN ARRIVE                                                   JOHN ARRIVE
  184: *IX BOY *GIVE1 TEACHER *MARY                                  ALL BOY GIVE TEACHER APPLE
  189: JOHN *JOHN *JOHN *CAN                                         JOHN GIVE GIRL BOX
  193: JOHN *IX *IX BOX                                              JOHN GIVE GIRL BOX
  199: *JOHN CHOCOLATE WHO                                           LIKE CHOCOLATE WHO
  201: JOHN *EAT MARY *JOHN BUY HOUSE                                JOHN TELL MARY IX-1P BUY HOUSE
```

Question 3: Summarize the error results from three combinations of features and model selectors. What was the "best" combination and why? What additional information might we use to improve our WER? For more insight on improving WER, take a look at the introduction to Part 4.

Answer 3: After running all the combinations of features and model selectors, I was not surprised my customized feature returned the best result for all the model selectors. But I was really impressed the constant model selector return the lowest WER of only 0.48. That's only 3 states in the HMM! Besides the SLM mentioned in the part 4. More optimized features may also worth the time to explore.

Recognizer Unit Tests

Run the following unit tests as a sanity check on the defined recognizer. The test simply looks for some valid values but is not exhaustive. However, the project should not be submitted if these tests don't pass.

In [34]:
```
from asl_test_recognizer import TestRecognize
suite = unittest.TestLoader().loadTestsFromModule(TestRecognize())
unittest.TextTestRunner().run(suite)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 36.695s

OK
```

Out[34]:
```
<unittest.runner.TextTestResult run=2 errors=0 failures=0>
```

PART 4: (OPTIONAL) Improve the WER with Language Models

We've squeezed just about as much as we can out of the model and still only get about 50% of the words right! Surely we can do better than that. Probability to the rescue again in the form of statistical language models (SLM) (https://en.wikipedia.org/wiki/Language_model). The basic idea is that each word has some probability of occurrence within the set, and some probability that it is adjacent to specific other words. We can use that additional information to make better choices.

Additional reading and resources

- Introduction to N-grams (Stanford Jurafsky slides) (https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf)
- Speech Recognition Techniques for a Sign Language Recognition System, Philippe Dreuw et al (https://www-i6.informatik.rwth-aachen.de/publications/download/154/Dreuw--2007.pdf) see the improved results of applying LM on *this* data!
- SLM data for *this* ASL dataset (ftp://wasserstoff.informatik.rwth-aachen.de/pub/rwth-boston-104/lm/)

Optional challenge

The recognizer you implemented in Part 3 is equivalent to a "0-gram" SLM. Improve the WER with the SLM data provided with the data set in the link above using "1-gram", "2-gram", and/or "3-gram" statistics. The `probabilities` data you've already calculated will be useful and can be turned into a pandas DataFrame if desired (see next cell).
Good luck! Share your results with the class!

In [ ]:
```
# create a DataFrame of log likelihoods for the test word items
df_probs = pd.DataFrame(data=probabilities)
df_probs.head()
```