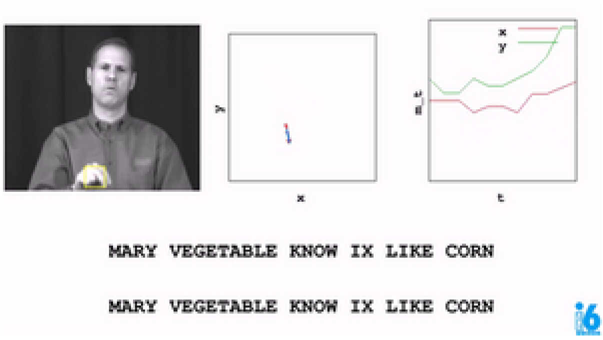
Artificial Intelligence Engineer Nanodegree - Probabilistic Models

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) to analyze a series of measurements taken from videos of American Sign Language (ASL) collected for research (see the RWTH-BOSTON-104 Database). In this video, the right-hand x and y locations are plotted as the speaker signs the sentence.



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.

[2]

```
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
```

		left-x	left-y	right-x	right-y	nose-x	nose-y	speake
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

[3]

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

```
F:\python\anaconda\lib\site-packages\ipykernel_launcher.py:1:
DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-
deprecated
  """Entry point for launching an IPython kernel.
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.

```
[4]  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 dictionar
```

		left-x	left-y	right-x	right-y	nose-x	nose-y	speake
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

Try it!

```
[5]  from asl_utils import test_features_tryit
     # TODO add df columns for 'grnd-rx', 'grnd-ly', 'grnd-lx' representing di
     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	speake
video	frame							
98	0	149	181	170	175	161	62	woman 1
	1	149	181	170	175	161	62	woman 1

		left-x	left-y	right-x	right-y	nose-x	nose-y	speake
video	frame							
	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

Correct!

[6]

```
# 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]
```

```
[9, 113, -12, 119]
```

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:

[7]

```
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', 'GO', 'GO1', 'FUTURE', 'GO2', 'PARTY',
'FUTURE1', 'HIT', 'BLAME', 'FRED', 'FISH', 'WONT', 'EAT', 'BUT',
'CHICKEN', 'VEGETABLE', 'CHINA', 'PEOPLE', 'PREFER', 'BROCCOLI', 'LIKE',
'LEAVE', 'SAY', 'BUY', 'HOUSE', 'KNOW', 'CORN', 'CORN1', 'THINK', 'NOT',
'PAST', 'LIVE', 'CHICAGO', 'CAR', 'SHOULD', 'DECIDE', 'VISIT', 'MOVIE',
'WANT', 'SELL', 'TOMORROW', 'NEXT-WEEK', 'NEW-YORK', 'LAST-WEEK', 'WILL',
'FINISH', 'ANN', 'READ', 'BOOK', 'CHOCOLATE', 'FIND', 'SOMETHING-ONE',
'POSS', 'BROTHER', 'ARRIVE', 'HERE', 'GIVE', 'MAN', 'NEW', 'COAT',
'WOMAN', 'GIVE1', 'HAVE', 'FRANK', 'BREAK-DOWN', 'SEARCH-FOR', 'WHO',
'WHAT', 'LEG', 'FRIEND', 'CANDY', 'BLUE', 'SUE', 'BUY1', 'STOLEN', 'OLD',
'STUDENT', 'VIDEOTAPE', 'BORROW', 'MOTHER', 'POTATO', 'TELL', 'BILL',
'THROW', 'APPLE', 'NAME', 'SHOOT', 'SAY-1P', 'SELF', 'GROUP', 'JANA',
```

```
'TOY1', 'MANY', 'TOY', 'ALL', 'BOY', 'TEACHER', '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).

[8]

```
training.get_word_sequences('WRITE')
```

```
[[[-20, 27, -10, 102],
  [-16, 31, -6, 92],
  [-12, 33, 0, 80],
  [-8, 40, 2, 71],
  [-3, 47, 6, 66],
  [-2, 49, 9, 60],
  [-7, 63, 10, 54],
  [-11, 60, 11, 55],
  [-12, 58, 10, 58],
  [-17, 55, 12, 57],
  [-17, 51, 11, 58],
  [-15, 49, 9, 58],
  [-11, 47, 10, 59],
  [-5, 44, 8, 57],
  [-3, 47, 10, 55],
  [1, 53, 9, 55],
  [-8, 54, 8, 56],
  [-12, 55, 8, 57],
  [-16, 52, 11, 59],
  [-21, 50, 11, 58],
  [-18, 45, 14, 57],
  [-17, 45, 11, 60],
  [-11, 45, 10, 60],
  [-6, 49, 8, 58],
  [-2, 55, 10, 57],
  [-5, 59, 8, 59],
  [-13, 58, 12, 58],
  [-17, 55, 10, 61],
  [-26, 51, 7, 60]]]
```

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 functions and pandas groupby. Below is an example for finding the means of all speaker subgroups.

```
[9]   df_means = asl.df.groupby('speaker').mean()
      df_means
```

	left-x	left-y	right-x	right-y	nose-x
speaker					
man-1	206.248203	218.679449	155.464350	150.371031	175.031756
woman-1	164.661438	161.271242	151.017865	117.332462	162.655120
woman-2	183.214509	176.527232	156.866295	119.835714	170.318973

To select a mean that matches by speaker, use the pandas map method:

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

		left-x	left-y	right-x	right-y	nose-x	nose-y	speake
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

Try it!

```
[11] from asl_utils import test_std_tryit
# TODO Create a dataframe named `df_std` with standard deviations grouped
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	nos
speaker						
man-1	15.154425	36.328485	18.901917	54.902340	6.654573	5.5
woman-1	17.573442	26.594521	16.459943	34.667787	3.549392	3.5
woman-2	15.388711	28.825025	14.890288	39.649111	4.099760	3.4

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 equation to account for speakers with different heights and arm length
- polar coordinates
  - calculate polar coordinates with Cartesian to polar equations
  - use the np.arctan2 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 and fillna method 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]
  - normalize the polar coordinates
  - adding additional deltas

[12]

```
# 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
asl.df['mean_lx'] = asl.df['speaker'].map(df_means['left-x'])
asl.df['std_lx'] = asl.df['speaker'].map(df_std['left-x'])

asl.df['mean_rx'] = asl.df['speaker'].map(df_means['right-x'])
asl.df['std_rx'] = asl.df['speaker'].map(df_std['right-x'])

asl.df['mean_ly'] = asl.df['speaker'].map(df_means['left-y'])
asl.df['std_ly'] = asl.df['speaker'].map(df_std['left-y'])

asl.df['mean_ry'] = asl.df['speaker'].map(df_means['right-y'])
asl.df['std_ry'] = asl.df['speaker'].map(df_std['right-y'])

asl.df['norm-rx']= (asl.df['right-x'] - asl.df['mean_rx']) / asl.df['std_
asl.df['norm-ry']= (asl.df['right-y'] - asl.df['mean_ry']) / asl.df['std_
asl.df['norm-lx']= (asl.df['left-x'] - asl.df['mean_lx']) / asl.df['std_l
asl.df['norm-ly']= (asl.df['left-y'] - asl.df['mean_ly']) / asl.df['std_l
features_norm = ['norm-rx', 'norm-ry', 'norm-lx','norm-ly']
```

[13]

```
# TODO add features for polar coordinate values where the nose is the ori
# Name these 'polar-rr', 'polar-rtheta', 'polar-lr', and 'polar-ltheta'
# Note that 'polar-rr' and 'polar-rtheta' refer to the radius and angle
asl.df['polar-rr'] = np.hypot(asl.df['right-x'] - asl.df['nose-x'],asl.df
asl.df['polar-lr'] = np.hypot(asl.df['left-x'] - asl.df['nose-x'],asl.df[
asl.df['polar-rtheta'] = np.arctan2(asl.df['right-x'] - asl.df['nose-x'],
asl.df['polar-ltheta'] = np.arctan2(asl.df['left-x'] - asl.df['nose-x'],a

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

[14]

```
# TODO add features for left, right, x, y differences by one time step, i
# Name these 'delta-rx', 'delta-ry', 'delta-lx', and 'delta-ly'
asl.df['delta-rx'] = asl.df['right-x'].diff().fillna(value=0)
asl.df['delta-ry'] = asl.df['right-y'].diff().fillna(value=0)
asl.df['delta-lx'] = asl.df['left-x'].diff().fillna(value=0)
asl.df['delta-ly'] = asl.df['left-y'].diff().fillna(value=0)
features_delta = ['delta-rx', 'delta-ry', 'delta-lx', 'delta-ly']
```

```
[15] # TODO add features of your own design, which may be a combination of the
     asl.df['delta-norm-rx'] = asl.df['norm-rx'].diff().fillna(value=0)
     asl.df['delta-norm-ry'] = asl.df['norm-ry'].diff().fillna(value=0)
     asl.df['delta-norm-lx'] = asl.df['norm-lx'].diff().fillna(value=0)
     asl.df['delta-norm-ly'] = asl.df['norm-ly'].diff().fillna(value=0)
     features_delta_norm = ['delta-norm-rx', 'delta-norm-ry', 'delta-norm-lx',

     features_custom = features_delta_norm + features_norm

     # TODO define a list named 'features_custom' for building the training se
```

```
[16] features_custom_notnorm = features_delta + features_ground
```

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

Answer 1: I build two simples custom features. As features_delta represent the speed, I'm using both the position and the speed of the hands to have a quite complete set of features. I don't know yep if the normalization is helpfull so I build the two kinds of features. One with normalization, the other without.

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.

```
[17] 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,

         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

         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]], "S

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

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.013s

OK
<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 (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.

[18]
```
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(
    print('lengths', lengths)
    print(X)
```

```
    print('sum', np.sum(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, m
print("logL = {}".format(logL))
```

```
 [  2 111  19 119]
 [  2 111  19 119]
 [  1  45  26  60]
 [  3  52  22  59]
 [  3  52  20  59]
 [  3  52  20  59]
 [  4  59  20  60]
 [  4  59  20  60]
 [  6  67  20  68]
 [  6  77  20  76]
 [ -7  36  15  42]
 [ -8  43  12  47]
 [ -8  43  12  47]
 [ -9  53  13  55]
 [-11  54  11  56]
 [-11  60  10  60]
 [-11  60  10  60]
 [ -7  65  10  65]
 [ -6  70  13  73]
 [ -7  76  14  80]
 [-16  77  24 105]
 [-17  75  24  96]
 [-13  82  24  85]
 [-13  80  20  84]
 [ -9  79  16  83]
 [ -9  79  16  83]
 [ -9  79  21  91]
 [ -9  79  17 105]]
sum 172
Number of states trained in model for BOOK is 3
```

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 for any individual sample or group of samples can also be calculated with the `score` method.

[19]

```
def show_model_stats(word, model):
    print("Number of states trained in model for {} is {}".format(word, m
    variance=np.array([np.diag(model.covars_[i]) for i in range(model.n_c
    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.

```
[20]
my_testword = 'CHOCOLATE'
model, logL = train_a_word(my_testword, 3, features_custom ) # Experiment
show_model_stats(my_testword, model)
print("logL = {}".format(logL))
```

```
 [ 0.          0.          0.          0.          0.4241895
0.01925529
   0.1899777  -0.87503896]
 [ 0.          0.          0.          0.          0.4241895
0.01925529
   0.1899777  -0.87503896]
 [ 0.18226065  0.20191655  0.          0.          0.60645015
0.22117184
   0.1899777  -0.87503896]
 [ 0.          0.          0.          0.          0.60645015
0.22117184
   0.1899777  -0.87503896]
 [ 0.1215071   0.08653567  0.1138081   0.26321211  0.72795725  0.3077075
   0.3037858  -0.61182685]
 [-0.06075355  0.17307133 -0.51213644  0.22561038  0.6672037
0.48077883
  -0.20835064 -0.38621647]
 [ 0.31742812 -0.0910708  -0.26394931 -0.38537253  0.18705246
-0.00675801
```

```
 -1.27013746 -0.8720278 ]
[ 0.47614218 -0.0910708  -0.06598733 -0.19268626  0.66319464
-0.09782881
 -1.33612479 -1.06471407]
[ 0.21161875 -0.07285664  0.13197466 -0.27526609  0.87481338
-0.17068545
 -1.20415013 -1.33998016]
[ 0.3703328   0.01821416 -0.26394931 -0.27526609  1.24514618
-0.15247129
 -1.46809944 -1.61524625]
[-0.05290469 -0.07285664 -0.19796199 -0.08257983  1.1922415
```

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.

[21]

```
%matplotlib inline
```

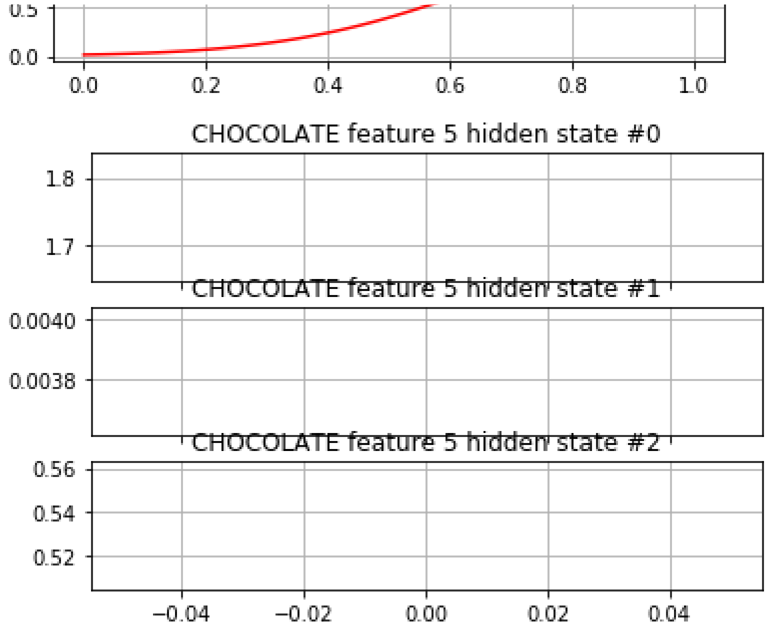
[22]

```
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_c
    figures = []
    for parm_idx in range(len(model.means_[0])):
        xmin = int(min(model.means_[:,parm_idx]) - max(variance[:,parm_id
        xmax = int(max(model.means_[:,parm_idx]) + max(variance[:,parm_id
        fig, axs = plt.subplots(model.n_components, sharex=True, sharey=F
        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, pa

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

visualize(my_testword, model)
```



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:

```
[23]  from my_model_selectors import SelectorConstant
```

```
training = asl.build_training(features_delta)  # Experiment here with dif
word = 'VEGETABLE' # Experiment here with different words
model = SelectorConstant(training.get_all_sequences(), training.get_all_X
print("Number of states trained in model for {} is {}".format(word, model
```

```
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 class object. When you implement `SelectorCV`, you will use this technique.

[24]
```
from sklearn.model_selection import KFold

training = asl.build_training(features_delta) # Experiment here with diff
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_id
```

```
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 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 (also found here). 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.

[25]
```
words_to_train = ['FISH', 'BOOK', 'VEGETABLE', 'FUTURE', 'JOHN']
import timeit
```

[26]
```
%load_ext autoreload
%autoreload 2

# TODO: Implement SelectorCV in my_model_selector.py
from my_model_selectors import SelectorCV

training = asl.build_training(features_delta)  # Experiment here with dif
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
    end = timeit.default_timer()-start
    if model is not None:
        print("Training complete for {} with {} states with time {} secon
    else:
        print("Training failed for {}".format(word))
```

```
Training complete for FISH with 3 states with time 0.4296131088810059
seconds
Training complete for BOOK with 15 states with time 4.606572423958194
seconds
Training complete for VEGETABLE with 15 states with time
1.6572931726341178 seconds
Training complete for FUTURE with 10 states with time 4.193376093993526
```

```
seconds
Training complete for JOHN with 11 states with time 37.798171329435746
seconds
```

[27]

```python
# TODO: Implement SelectorBIC in module my_model_selectors.py
from my_model_selectors import SelectorBIC

training = asl.build_training(features_delta)  # Experiment here with dif
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
    end = timeit.default_timer()-start
    if model is not None:
        print("Training complete for {} with {} states with time {} secon
    else:
        print("Training failed for {}".format(word))
```

```
Training complete for FISH with 4 states with time 0.3434339231436496
seconds
Training complete for BOOK with 8 states with time 2.3672736376651144
seconds
Training complete for VEGETABLE with 3 states with time 0.732204842133477
seconds
Training complete for FUTURE with 5 states with time 1.64352666135391
seconds
Training complete for JOHN with 6 states with time 19.377283017271168
seconds
```

[28]

```python
# TODO: Implement SelectorDIC in module my_model_selectors.py
from my_model_selectors import SelectorDIC

training = asl.build_training(features_delta)  # Experiment here with dif
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
    end = timeit.default_timer()-start
    if model is not None:
        print("Training complete for {} with {} states with time {} secon
    else:
        print("Training failed for {}".format(word))
```

```
first generation of dict_logL
Training complete for FISH with 4 states with time 51.53774268849129
seconds
```

```
Training complete for BOOK with 9 states with time 0.1451063844792202
seconds
Training complete for VEGETABLE with 3 states with time
0.08470031100799247 seconds
Training complete for FUTURE with 5 states with time 0.09864454113980514
seconds
Training complete for JOHN with 10 states with time 1.8458831992214186
seconds
```

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

Answer 2: If we compare the speed of calculation the SelectorBIC seems the best right now. But if we take into account that we will train for all the words our models SelectorDIC will be even quicker. If we compare the number of states obtained, the selectorCV seems quite different that the other two. With on average many more states found. It seems unlikely that we need as many states to describe a movement as those found in SelectorCV.

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.

[46]

```python
from asl_test_model_selectors import TestSelectors
suite = unittest.TestLoader().loadTestsFromModule(TestSelectors())
unittest.TextTestRunner().run(suite)
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 49.709s

OK
<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.

[47]
```
# autoreload for automatically reloading changes made in my_model_selecto
%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 diffe
    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)))
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
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`

[31]
```
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_in
```

```
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.

[32]
```
# TODO implement the recognize method in my_recognizer
from my_recognizer import recognize
from asl_utils import show_errors
```

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

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

```
WOMAN ARRIVE
  113: IX CAR *CAR *IX *IX                                    IX
CAR BLUE SUE BUY
  119: *PREFER *BUY1 IX *BLAME *IX                            SUE
BUY IX CAR BLUE
  122: JOHN *GIVE1 *COAT
JOHN READ BOOK
  139: *SHOULD *BUY1 *CAR *BLAME BOOK
JOHN BUY WHAT YESTERDAY BOOK
  142: *FRANK *STUDENT YESTERDAY *TEACHER BOOK
JOHN BUY YESTERDAY WHAT BOOK
  158: LOVE *MARY WHO
LOVE JOHN WHO
  167: *MARY IX *VISIT *WOMAN *LOVE
JOHN IX SAY LOVE MARY
  171: *VISIT *VISIT BLAME
JOHN MARY BLAME
  174: *CAN *GIVE3 GIVE1 *APPLE *WHAT
PEOPLE GROUP GIVE1 JANA TOY
  181: *BLAME ARRIVE
JOHN ARRIVE
  184: *GIVE1 BOY *GIVE1 TEACHER APPLE                        ALL
```

BOY GIVE TEACHER APPLE
  189: *JANA *SOMETHING-ONE *YESTERDAY *WHAT
JOHN GIVE GIRL BOX
  193: JOHN *SOMETHING-ONE *YESTERDAY BOX
JOHN GIVE GIRL BOX
  199: *LOVE CHOCOLATE WHO
LIKE CHOCOLATE WHO

[34]

```
# TODO Choose a feature set and model selector
features = features_custom_notnorm # change as needed
model_selector = SelectorConstant # change as needed
# TODO Recognize the test set and display the result with the show_errors
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 = 0.47752808988764045
Total correct: 93 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
```

[35]

```
# TODO Choose a feature set and model selector
features = features_custom_notnorm # change as needed
```

```
model_selector = SelectorBIC # change as needed
# TODO Recognize the test set and display the result with the show_errors
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 = 0.5112359550561798
Total correct: 87 out of 178
Video  Recognized
Correct
=========================================================================
=============================
    2: JOHN *NEW *ARRIVE
JOHN WRITE HOMEWORK
    7: JOHN *CAR *IX *JOHN
JOHN CAN GO CAN
   12: JOHN CAN *WHAT CAN
JOHN CAN GO CAN
   21: JOHN *MARY *JOHN *WHO *CAR *CAR *ARRIVE *BOOK
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN *MARY *MARY IX *MARY
JOHN LIKE IX IX IX
   28: JOHN *WHO *MARY *JOHN IX
JOHN LIKE IX IX IX
   30: JOHN LIKE *MARY IX IX
JOHN LIKE IX IX IX
   36: MARY *JOHN *JOHN IX *MARY *MARY
MARY VEGETABLE KNOW IX LIKE CORN1
   40: JOHN IX *JOHN MARY *MARY
JOHN IX THINK MARY LOVE
   43: JOHN *SHOULD BUY HOUSE
JOHN MUST BUY HOUSE
   50: *JOHN *MARY BUY CAR SHOULD
FUTURE JOHN BUY CAR SHOULD
   54: JOHN *FUTURE *FUTURE BUY HOUSE
```

[36]

```
# TODO Choose a feature set and model selector
features = features_custom_notnorm # change as needed
model_selector = SelectorCV # change as needed
# TODO Recognize the test set and display the result with the show_errors
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 = 0.5168539325842697
Total correct: 86 out of 178
```

```
Video  Recognized
Correct
=========================================================================
=============================
    2: JOHN *NEW *ARRIVE
JOHN WRITE HOMEWORK
    7: JOHN *CAR *IX *JOHN
JOHN CAN GO CAN
   12: JOHN *WHAT *WHAT CAN
JOHN CAN GO CAN
   21: JOHN *JOHN *JOHN *MARY *WHAT *CAR *ARRIVE *WHO
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: *MARY *JOHN *JOHN IX *JOHN
JOHN LIKE IX IX IX
   28: JOHN *WHO IX *JOHN IX
JOHN LIKE IX IX IX
   30: JOHN *MARY *MARY IX IX
JOHN LIKE IX IX IX
   36: MARY *JOHN *IX IX *MARY *MARY
MARY VEGETABLE KNOW IX LIKE CORN1
   40: JOHN IX *JOHN MARY *MARY
JOHN IX THINK MARY LOVE
   43: JOHN *JOHN BUY HOUSE
JOHN MUST BUY HOUSE
   50: *MARY JOHN BUY CAR *WHAT
FUTURE JOHN BUY CAR SHOULD
   54: JOHN *FUTURE *FUTURE BUY HOUSE
```

[37]

```
# TODO Choose a feature set and model selector
features = features_custom_notnorm # change as needed
model_selector = SelectorDIC # change as needed
# TODO Recognize the test set and display the result with the show_errors
models = train_all_words(features, model_selector)
test_set = asl.build_test(features)
probabilities, guesses = recognize(models, test_set)
show_errors(guesses, test_set)
```

```
first generation of dict_logL

**** WER = 0.5
Total correct: 89 out of 178
Video  Recognized
Correct
=========================================================================
=============================
    2: JOHN *NEW *ARRIVE
JOHN WRITE HOMEWORK
    7: JOHN *CAR *IX *JOHN
JOHN CAN GO CAN
   12: JOHN CAN *WHAT CAN
JOHN CAN GO CAN
```

```
   21: JOHN *JOHN *JOHN *MARY *CAR *CAR *ARRIVE *WHO
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: *MARY *JOHN *JOHN IX *JOHN
JOHN LIKE IX IX IX
   28: JOHN *WHO IX *JOHN IX
JOHN LIKE IX IX IX
   30: JOHN *MARY *MARY IX IX
JOHN LIKE IX IX IX
   36: MARY *JOHN *IX IX *MARY *MARY
MARY VEGETABLE KNOW IX LIKE CORN1
   40: JOHN IX *JOHN MARY *MARY
JOHN IX THINK MARY LOVE
   43: JOHN *JOHN BUY HOUSE
JOHN MUST BUY HOUSE
   50: *MARY JOHN BUY CAR *JOHN
FUTURE JOHN BUY CAR SHOULD
```

[38]

```python
# TODO Choose a feature set and model selector
all_features = {'features_ground':features_ground, 'features_norm':featur
                'features_polar':features_polar, 'features_delta':feature
                'features_custom':features_custom, 'features_custom_notno
model_selector = SelectorDIC # change as needed
for features_names,features in all_features.items() :
# TODO Recognize the test set and display the result with the show_errors
    models = train_all_words(features, model_selector)
    test_set = asl.build_test(features)
    probabilities, guesses = recognize(models, test_set)
    print(features_names)
    show_errors(guesses, test_set)
```

```
first generation of dict_logL
features_ground

**** WER = 0.5786516853932584
Total correct: 75 out of 178
Video  Recognized
Correct
=========================================================================
=============================
    2: JOHN *NEW *GIVE1
JOHN WRITE HOMEWORK
    7: *SOMETHING-ONE *CAR *TOY1 *TOY
JOHN CAN GO CAN
   12: *IX *WHAT *WHAT *CAR
JOHN CAN GO CAN
   21: JOHN *GIVE1 *JOHN *FUTURE *NEW-YORK *CAR *CHICAGO *MARY
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN *IX IX *WHO IX
JOHN LIKE IX IX IX
   28: JOHN *WHO IX IX *LOVE
JOHN LIKE IX IX IX
```

```
   30: JOHN *MARY *MARY *MARY *MARY
JOHN LIKE IX IX IX
   36: *VISIT *VISIT *IX *GO *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
```

[39]

```
# TODO Choose a feature set and model selector
features = features_custom # change as needed
model_selector = SelectorDIC # change as needed
# TODO Recognize the test set and display the result with the show_errors
models = train_all_words(features, model_selector)
test_set = asl.build_test(features)
probabilities, guesses = recognize(models, test_set)
show_errors(guesses, test_set)
```

```
[autoreload of my_model_selectors failed: Traceback (most recent call
last):
  File "F:\python\anaconda\lib\site-
packages\IPython\extensions\autoreload.py", line 246, in check
    superreload(m, reload, self.old_objects)
  File "F:\python\anaconda\lib\site-
packages\IPython\extensions\autoreload.py", line 369, in superreload
    module = reload(module)
  File "F:\python\anaconda\lib\imp.py", line 315, in reload
    return importlib.reload(module)
  File "F:\python\anaconda\lib\importlib\__init__.py", line 166, in
reload
    _bootstrap._exec(spec, module)
  File "<frozen importlib._bootstrap>", line 618, in _exec
  File "<frozen importlib._bootstrap_external>", line 674, in
exec_module
  File "<frozen importlib._bootstrap_external>", line 781, in get_code
  File "<frozen importlib._bootstrap_external>", line 741, in
source_to_code
  File "<frozen importlib._bootstrap>", line 219, in
_call_with_frames_removed
  File "F:\git\aind\AIND-Recognizer\my_model_selectors.py", line 129
    def __init__(self, all_word_sequences: dict, all_word_Xlengths:
dict, this_word: str,

^
TabError: inconsistent use of tabs and spaces in indentation
]
first generation of dict_logL
```

[40]

```
# TODO Choose a feature set and model selector
```

```python
all_features = {'features_ground':features_ground, 'features_norm':featur
                'features_polar':features_polar, 'features_delta':feature
                'features_custom':features_custom, 'features_custom_notno
model_selector = SelectorConstant # change as needed
for features_names,features in all_features.items() :
# TODO Recognize the test set and display the result with the show_errors
    models = train_all_words(features, model_selector)
    test_set = asl.build_test(features)
    probabilities, guesses = recognize(models, test_set)
    print(features_names)
    show_errors(guesses, test_set)
```

```
features_ground

**** WER = 0.6685393258426966
Total correct: 59 out of 178
Video  Recognized
Correct
========================================================================
=============================
    2: *GO WRITE *ARRIVE
JOHN WRITE HOMEWORK
    7: *SOMETHING-ONE *GO1 *IX CAN
JOHN CAN GO CAN
   12: JOHN *HAVE *WHAT CAN
JOHN CAN GO CAN
   21: JOHN *HOMEWORK *NEW *PREFER *CAR *CAR *FUTURE *EAT
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: *FRANK *TELL *LOVE *TELL *LOVE
JOHN LIKE IX IX IX
   28: *FRANK *TELL *LOVE *TELL *LOVE
JOHN LIKE IX IX IX
   30: *SHOULD LIKE *GO *GO *GO
JOHN LIKE IX IX IX
   36: *VISIT VEGETABLE *YESTERDAY *GIVE *MARY *MARY
MARY VEGETABLE KNOW IX LIKE CORN1
   40: *SUE *GIVE *CORN *VEGETABLE *GO
JOHN IX THINK MARY LOVE
   43: *FRANK *GO BUY HOUSE
JOHN MUST BUY HOUSE
   50: *FRANK *SEE BUY CAR *SOMETHING-ONE
FUTURE JOHN BUY CAR SHOULD
```

```python
[41]  # TODO Choose a feature set and model selector
all_features = {'features_ground':features_ground, 'features_norm':featur
                'features_polar':features_polar, 'features_delta':feature
                'features_custom':features_custom, 'features_custom_notno
model_selector = SelectorBIC # change as needed
for features_names,features in all_features.items() :
# TODO Recognize the test set and display the result with the show_errors
    models = train_all_words(features, model_selector)
```

```
    test_set = asl.build_test(features)
    probabilities, guesses = recognize(models, test_set)
    print(features_names)
    show_errors(guesses, test_set)
```

```
features_ground

**** WER = 0.6348314606741573
Total correct: 65 out of 178
Video  Recognized
Correct
========================================================================
=============================
    2: JOHN *NEW *ARRIVE
JOHN WRITE HOMEWORK
    7: *SOMETHING-ONE CAN *IX *ARRIVE
JOHN CAN GO CAN
   12: JOHN *WHAT *CAN CAN
JOHN CAN GO CAN
   21: *MARY *GIVE1 *GIVE1 *FUTURE *CAR *CAR *CHICAGO *WHO
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN *IX *LOVE *WHO IX
JOHN LIKE IX IX IX
   28: *IX *WHO IX IX *LOVE
JOHN LIKE IX IX IX
   30: *IX *MARY IX *GO IX
JOHN LIKE IX IX IX
   36: *VISIT *VISIT *IX *GO *MARY *MARY
MARY VEGETABLE KNOW IX LIKE CORN1
   40: *MARY IX *GIVE MARY *IX
JOHN IX THINK MARY LOVE
   43: JOHN *FUTURE BUY HOUSE
JOHN MUST BUY HOUSE
   50: *GO *MARY BUY CAR *JOHN
FUTURE JOHN BUY CAR SHOULD
```

[42]

```
# TODO Choose a feature set and model selector
all_features = {'features_ground':features_ground, 'features_norm':featur
                'features_polar':features_polar, 'features_delta':feature
                'features_custom':features_custom, 'features_custom_notno
model_selector = SelectorCV # change as needed
for features_names,features in all_features.items() :
# TODO Recognize the test set and display the result with the show_errors
    models = train_all_words(features, model_selector)
    test_set = asl.build_test(features)
    probabilities, guesses = recognize(models, test_set)
    print(features_names)
    show_errors(guesses, test_set)
```

```
features_ground

**** WER = 0.5842696629213483
Total correct: 74 out of 178
Video  Recognized
Correct
================================================================================
=============================
    2: JOHN *NEW *GIVE1
JOHN WRITE HOMEWORK
    7: *SOMETHING-ONE *CAR *TOY1 *WHAT
JOHN CAN GO CAN
   12: *IX *WHAT *WHAT *CAR
JOHN CAN GO CAN
   21: JOHN *GIVE1 *JOHN *FUTURE *CAR *CAR *CHICAGO *MARY
JOHN FISH WONT EAT BUT CAN EAT CHICKEN
   25: JOHN *IX IX *WHO IX
JOHN LIKE IX IX IX
   28: JOHN *WHO IX *FUTURE *LOVE
JOHN LIKE IX IX IX
   30: JOHN LIKE *MARY *MARY *MARY
JOHN LIKE IX IX IX
   36: *VISIT *VISIT *IX *GO *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 JOHN *GIVE1 CAR *JOHN
FUTURE JOHN BUY CAR SHOULD
```

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:

| features | Model | WER
| :- |-------------: | :-: |features_ground| SelectorConstant | 0.67 |features_ground| SelectorDIC | 0.56 |features_ground| SelectorBIC | 0.63 |features_ground| SelectorCV | 0.58

| features | Model | WER
| :- |-------------: | :-: |features_custom_notnorm| SelectorConstant | 0.48 |features_custom_notnorm| SelectorDIC | 0.50 |features_custom_notnorm| SelectorBIC | 0.51 |features_custom_notnorm| SelectorCV | 0.52

| features | Model | WER
| :- |-------------: | :-: |features_custom| SelectorConstant | 0.52 |features_custom| SelectorDIC | 0.48 |features_custom| SelectorBIC | 0.51 |features_custom| SelectorCV | 0.50

the best combination of features et model selector seems to be features_custom_notnorm with SelectorConstant at equality with features_custom with SelectorDIC. Our discriminators doen't seem to work very well, probablmy because we are underfitting. We don't have enough features to work with. To improve this current result we can use more features and to assure to not overfitting we can use a PCA filtering. Another option is to take into account the previous word obtained in our guessing sentence. Due to the short lenght of our sentence 1-gram or 2-gram strategy will be usefull

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.

[52]

```
from asl_test_recognizer import TestRecognize
suite = unittest.TestLoader().loadTestsFromModule(TestRecognize())
unittest.TextTestRunner().run(suite)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 31.611s

OK
<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). 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)
- Speech Recognition Techniques for a Sign Language Recognition System, Philippe Dreuw et al see the improved results of applying LM on *this* data!
- SLM data for *this* ASL dataset

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!

[51]

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

	ALL	ANN	APPLE	ARRIVE	BILL
0	-6923.781663	-1000000	-73000.558927	-635.190804	-49454.77230
1	-10703.051861	-1000000	-92404.675711	-401.840446	-63941.79304
2	-17220.137602	-1000000	-191282.328418	-751.932305	-127647.9028
3	-5294.270701	-1000000	-8871.010711	-680.492259	-7873.754399
4	-4289.201888	-1000000	-148833.384392	-198.678444	-138869.3789

5 rows × 112 columns