Open in Colab

Project 3 - Name Gender Classifier

by Gracie Hui Han and Don Padmaperuma

Assignment

Using any of the three classifiers described in chapter 6 of Natural Language Processing with Python, and any features you can think of, build the best name gender classifier you can.

Begin by splitting the Names Corpus into three subsets: 500 words for the test set, 500 words for the dev-test set, and the remaining 6900 words for the training set.

Then, starting with the example name gender classifier, make incremental improvements.

Use the dev-test set to check your progress. Once you are satisfied with your classifier, check its final performance on the test set.

How does the performance on the test set compare to the performance on the dev-test set? Is this what you'd expect?

In [1]:
```
import collections
import nltk
nltk.download('names')
from nltk.corpus import names
from nltk.classify import apply_features
from nltk.metrics import ConfusionMatrix, accuracy, precision, recall, f_measure
import random
```

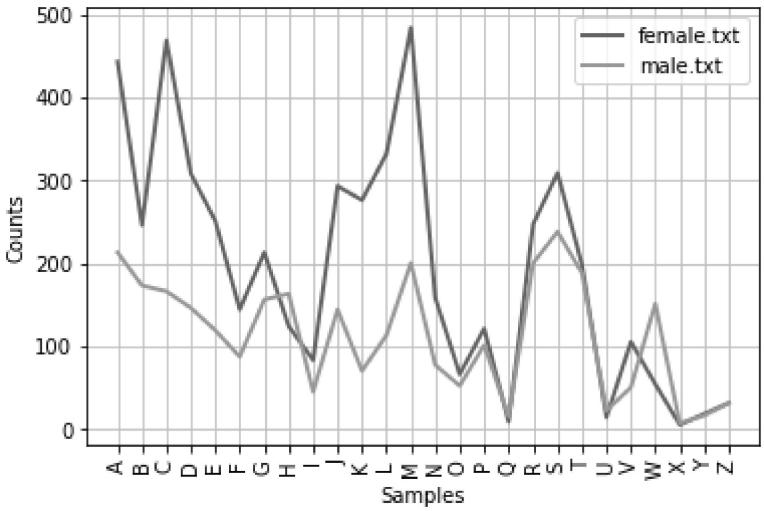
```
[nltk_data] Downloading package names to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package names is already up-to-date!
```

In [2]:
```
len(names.words('female.txt')), len(names.words('male.txt'))
```

Out[2]:
```
(5001, 2943)
```

We can see from above numbers that there are more female names (almost twise as male names) compared to the male names in the dataset.

In [3]:
```
# Check distribution of the first letter of the name.
gender_freq = nltk.ConditionalFreqDist((fileid, name[0])
  for fileid in names.fileids()
  for name in names.words(fileid))
gender_freq.plot()
```



Out[3]: <AxesSubplot:xlabel='Samples', ylabel='Counts'>

Above graph shows the comparison of distribution of beginning letter of the names by gender. We can see that more female names starts with letter A, C, M and S.

In [4]:
```
# combine male and female names and shuffle them.
names = ([(name, 'male') for name in names.words('male.txt')] + \
        [(name, 'female') for name in names.words('female.txt')])
random.shuffle(names)
```

In [5]:
```
# Take a look at the combined data
names[1:5]
```

Out[5]:
```
[('Sunny', 'female'),
 ('Shelley', 'male'),
 ('Deva', 'female'),
 ('Moll', 'female')]
```

In [6]:
```
# unique names
len(set(item[0] for item in names))
```

Out[6]: 7579

We removed the names that was listed as both males and females and was listed twice.

In [7]:
```
# names that are not unique
names_only = [item[0] for item in names]
names_dist = nltk.FreqDist(names_only)
names_duplicates = [(k,v) for k,v in names_dist.items() if v>1 ]
names_duplicates [0:10]
```

Out[7]:
```
[('Sunny', 2),
 ('Shelley', 2),
 ('Shea', 2),
 ('Augustine', 2),
 ('Nat', 2),
 ('Gail', 2),
 ('Gabriell', 2),
 ('Lou', 2),
 ('Danny', 2),
 ('Frankie', 2)]
```

In [8]:
```
# Remove the duplicates and show total number of unique names
names2remove = [item[0] for item in names_duplicates]
final_names = [item for item in names if not item[0] in names2remove]
len(final_names)
```

Out[8]: 7214

In [9]:
```
names = final_names
len(names)
```

Out[9]: 7214

There are 7214 Unique Names in this dataset

Split into Train, DevTest and Test Data

In [10]:
```
train_set = names[1000:]        # Training set
test_set = names[:500]   # Test
devtest_set = names[500:1000]   #development test
```

In [11]:
```
# Show size of the three subsets
print("Training Set = {}".format(len(train_set)))
print("Dev-Test Set = {}".format(len(devtest_set)))
print("Test Set = {}".format(len(test_set)))
```

```
Training Set = 6214
Dev-Test Set = 500
Test Set = 500
```

The builtin performance from the NLTK will only provide the overall performance all the data set, which does not differentiate male and female. Therefore, below function is created, which look at male/ female separately. for the obvious reason,a male and a female name performance should be different, at least this is what we expect. we will use this customize the function to Calculate the performance, which will be used to compare the built in performance of pooled male and female database.

In [12]:
```
# Write a function for manually calculate the recall error and precision formance, for
#(to compare the automatically generated performance indicator)

def performance_metrics(model, train, digits=4):
    """Prints the precision and recall of an NLTK Naive Bayes model."""

    reference = collections.defaultdict(set)
    test = collections.defaultdict(set)

    for i, (features, label) in enumerate(train):
        reference[label].add(i)
        pred = model.classify(features)
        test[pred].add(i)

    m_precision = round(precision(reference['male'], test['male']), digits)
    f_precision = round(precision(reference['female'], test['female']), digits)

    m_recall = round(recall(reference['male'], test['male']), digits)
    f_recall = round( recall(reference['female'], test['female']), digits)

    print('Male precision: ', m_precision)
```

```
    print('Female precision: ', f_precision)
    print('Male recall: ', m_recall)
    print('Female recall: ', f_recall)
```

Features - First Feature

First Letter, Last Letter, Suffix, Prefix, Vowels

The first feature contains the first letter, last letter, prefix and suffix, as well as the vowels (aeiou, we did not include the y in here which is optional)

There has been a number of references, which performs very extensive feature combos Undis database. Minija come up with more than 10th features, and we are not going to repeat these same 10 features for this project. rather we are working on comparing the performance as out focus.

Out of the 20 feature sets that we referenced , we decide to choose 3 features set

- First feature is a very expensive one. Based on reference, this extensive feature, which takes into consideration of first letter, last letter, suffix, prefix, and the vowels. This is by far one of the best features based on reference, one of the best features that people have overwhelmingly voted on.

Bing that the first feature is one of the most comprehensive one, the next 2 features we decide that we want to look at the most simplified feature, yet based on other peoples analysis, they really are very telling of the performance, despite thar they are simple features.

- Second feature, we look at the last letter only.
- Third feature, we looked at the last 2 letters, with the intention to see if there is any difference in Adding one more letter 2 the last letter only

In [13]:
```
def gender_features1(name):
    features = {}
    features["firstletter"] = name[0].lower()
    features["lastletter"] = name[-1].lower()
    features["suffix2"]= name[-2:].lower()
    features["preffix2"]= name[:2].lower()

    for letter in 'aeiou':   ## vowels
        features["count(%s)" % letter] = name.lower().count(letter)
        features["has(%s)" % letter] = (letter in name.lower())
    return features
```

In [14]:
```
featuresets = [(gender_features1(n), g) for (n,g) in names]
featuresets[0]
```

Out[14]:
```
({'firstletter': 'i',
  'lastletter': 'e',
  'suffix2': 'ne',
  'preffix2': 'id',
  'count(a)': 1,
  'has(a)': True,
  'count(e)': 1,
```

```
 'has(e)': True,
 'count(i)': 2,
 'has(i)': True,
 'count(o)': 0,
 'has(o)': False,
 'count(u)': 0,
 'has(u)': False},
'female')
```

Split Data Based on 1st Feature

In [15]:
```
### Split data
train_set_fe = featuresets[1000:]
test_set_fe =featuresets[:500]
devtest_set_fe =featuresets[500:1000]
```

Classifier - NaiveBayes -on 1st Feature

In [16]:
```
classifier = nltk.NaiveBayesClassifier.train(train_set_fe)
print(classifier.classify(gender_features1('Stefani')))
print(classifier.classify(gender_features1('Steffan')))
```

```
female
male
```

In [17]:
```
# Show Accuracy
print("train_set: ", nltk.classify.accuracy(classifier, train_set_fe))
print("test_set: ", nltk.classify.accuracy(classifier, test_set_fe))
print("devtest_set: ", nltk.classify.accuracy(classifier, devtest_set_fe))
```

```
train_set:  0.8393949147087223
test_set:  0.854
devtest_set:  0.866
```

In [18]:
```
# Show important features
classifier.show_most_informative_features(5)
```

```
Most Informative Features
                suffix2 = 'na'           female : male   =    148.5 : 1.0
                suffix2 = 'la'           female : male   =     66.1 : 1.0
             lastletter = 'k'              male : female =     63.8 : 1.0
             lastletter = 'a'           female : male   =     58.8 : 1.0
                suffix2 = 'us'             male : female =     57.7 : 1.0
```

In [19]:
```
# Check errors
errors = []
for (name, tag) in devtest_set:
    guess = classifier.classify(gender_features1(name))
    if guess != tag:
        errors.append( (tag, guess, name) )
```

In [20]:
```
for (tag, guess, name) in sorted(errors):
   print('correct=%-8s guess=%-8s name=%-30s' % (tag, guess, name))
```

```
correct=female   guess=male     name=Aggy
correct=female   guess=male     name=Avis
correct=female   guess=male     name=Chloe
correct=female   guess=male     name=Corliss
correct=female   guess=male     name=Cybill
correct=female   guess=male     name=Donnajean
correct=female   guess=male     name=Dorey
```

```
correct=female   guess=male     name=Dotty
correct=female   guess=male     name=Dusty
correct=female   guess=male     name=Edy
correct=female   guess=male     name=Eleanor
correct=female   guess=male     name=Farrand
correct=female   guess=male     name=Gaby
correct=female   guess=male     name=Glory
correct=female   guess=male     name=Gunvor
correct=female   guess=male     name=Hatty
correct=female   guess=male     name=Helen
correct=female   guess=male     name=Jackquelin
correct=female   guess=male     name=Jeniffer
correct=female   guess=male     name=Jennifer
correct=female   guess=male     name=Jo Ann
correct=female   guess=male     name=Joscelin
correct=female   guess=male     name=Joyann
correct=female   guess=male     name=Laurel
correct=female   guess=male     name=Lorrin
correct=female   guess=male     name=Marj
correct=female   guess=male     name=Mildred
correct=female   guess=male     name=Nan
correct=female   guess=male     name=Noellyn
correct=female   guess=male     name=Peggy
correct=female   guess=male     name=Poppy
correct=female   guess=male     name=Renel
correct=female   guess=male     name=Rosario
correct=female   guess=male     name=Roselin
correct=female   guess=male     name=Ruthanne
correct=female   guess=male     name=Ruthe
correct=female   guess=male     name=Scarlet
correct=female   guess=male     name=Sharon
correct=female   guess=male     name=Ursuline
correct=female   guess=male     name=Winonah
correct=male     guess=female   name=Adolphe
correct=male     guess=female   name=Ashby
correct=male     guess=female   name=Bealle
correct=male     guess=female   name=Carl
correct=male     guess=female   name=Chaddie
correct=male     guess=female   name=Chrisy
correct=male     guess=female   name=Clancy
correct=male     guess=female   name=Costa
correct=male     guess=female   name=Davide
correct=male     guess=female   name=Enrique
correct=male     guess=female   name=Hiralal
correct=male     guess=female   name=Jay
correct=male     guess=female   name=Jerri
correct=male     guess=female   name=Kalman
correct=male     guess=female   name=Lay
correct=male     guess=female   name=Lazare
correct=male     guess=female   name=Mendie
correct=male     guess=female   name=Merrel
correct=male     guess=female   name=Merwin
correct=male     guess=female   name=Nikita
correct=male     guess=female   name=Ole
correct=male     guess=female   name=Partha
correct=male     guess=female   name=Prentice
correct=male     guess=female   name=Siffre
correct=male     guess=female   name=Slade
correct=male     guess=female   name=Stanly
correct=male     guess=female   name=Verne
```

In [21]:
```
print("Error count: ", len(errors))
```

```
Error count:  67
```

In [22]:
```
train_set1 = [(gender_features1(n), g) for (n,g) in train_set]
devtest_set1 = [(gender_features1(n), g) for (n,g) in devtest_set]
test_set1 = [(gender_features1(n), g) for (n,g) in test_set]

nb1 = nltk.NaiveBayesClassifier.train(train_set1)

print('Devset accuracy is')
print(nltk.classify.accuracy(nb1, devtest_set1))
print('Test accuracy is')
print(nltk.classify.accuracy(nb1, test_set1))
```

```
Devset accuracy is
0.866
Test accuracy is
0.854
```

In [23]:
```
performance_metrics(nb1, train_set1)
```

```
Male precision:  0.7601
Female precision:  0.8879
Male recall:  0.8058
Female recall:  0.8581
```

In [24]:
```
performance_metrics(nb1, devtest_set1)
```

```
Male precision:  0.7849
Female precision:  0.914
Male recall:  0.8439
Female recall:  0.8777
```

We have found that that tough as expected, this extensive feature give us a satisfying accuracy on both test set and the dev test set.

- and the performance on deaf test and the test are similar.
- using our customized function, we looked at the performance of male and female separatedly
- As expected come up because this feature is such an extensive one, which takes into consideration of every possible scenario, the performance a male and a female are similar.
- On top of that, the performance for the pooled may on the female set, are similar to male set only, and female set only.

Classifier - DecisionTree-on 1st Feature

In [25]:
```
classifier_tree = nltk.DecisionTreeClassifier.train(train_set_fe)

print("train_set: ", nltk.classify.accuracy(classifier_tree, train_set_fe))
print("test_set: ", nltk.classify.accuracy(classifier_tree, test_set_fe))
print("devtest_set: ", nltk.classify.accuracy(classifier_tree, devtest_set_fe))
```

```
train_set:  0.9765046668812359
test_set:  0.832
devtest_set:  0.844
```

The decision tree classifier accuracy are similar to the bayesians model accuracy, using this first feature, which is an extensive feature

Feature2 -First Letter Only

The next 2 features, we switched direction totale, from looking at the most extensive feature, to looking at the least expensive feature. because there are unlimited numbers of features that we can look at, we read other references extensively, and the omitted all these importunes ones, and jumped into using the least comprehensive, yet most telling one, out of the possible features period

- Based on reference, the first letter alone and the last 2 letters, are much more important than the rest of the inbtween letters.
- So lets test these last letter has the second feature, and the last 2 letters as the third

In [ ]:
```
```

In [26]:
```
def first_letter(name):
    name = name.lower()
    return {
        'first_1_letter': name[0]
        }
first_letter("Mary")
```

Out[26]:
```
{'first_1_letter': 'm'}
```

In [27]:
```
train_set2 = [(first_letter(n), g) for (n,g) in train_set] ###
devtest_set2 = [(first_letter(n), g) for (n,g) in devtest_set] ###
test_set2 = [(first_letter(n), g) for (n,g) in test_set] ###

nb1b = nltk.NaiveBayesClassifier.train(train_set2)  ###

print('Devset accuracy is')
print(nltk.classify.accuracy(nb1b, devtest_set2)) ###
print('Test accuracy is')
print(nltk.classify.accuracy(nb1b, test_set2)) ###
```

```
Devset accuracy is
0.662
Test accuracy is
0.66
```

In [28]:
```
performance_metrics(nb1b, devtest_set2) ###
```

```
Male precision:  0.54
Female precision:  0.6756
Male recall:  0.1561
Female recall:  0.9297
```

Very interestingly come on we want to point out that even this is such an extensive feature, it performs very well on female name.

Female name have a recall of 0.96, meaning the false positive rate is only less than 4%

However, the mail recall is 0.14, meaning that the recall error is as large as 95%

This is by far the most striking difference that we have found.

In [29]:
```
performance_metrics(nb1b, train_set2) ###
```

```
Male precision:  0.634
Female precision:  0.6661
Male recall:  0.1425
Female recall:  0.9541
```

In [30]:
```
# Looking at the most informative features
nb1b.show_most_informative_features(10)  ###
```

```
Most Informative Features
          first_1_letter = 'w'              male : female =      6.1 : 1.0
          first_1_letter = 'u'              male : female =      3.3 : 1.0
          first_1_letter = 'q'              male : female =      2.7 : 1.0
          first_1_letter = 'k'            female : male   =      2.4 : 1.0
          first_1_letter = 'h'              male : female =      2.4 : 1.0
          first_1_letter = 'z'              male : female =      2.0 : 1.0
          first_1_letter = 'l'            female : male   =      1.9 : 1.0
          first_1_letter = 'y'              male : female =      1.9 : 1.0
          first_1_letter = 'c'            female : male   =      1.9 : 1.0
          first_1_letter = 'x'              male : female =      1.8 : 1.0
```

In [31]:
```
classifier_tree = nltk.DecisionTreeClassifier.train(train_set2)

print("train_set: ", nltk.classify.accuracy(classifier_tree, train_set2))
print("test_set: ", nltk.classify.accuracy(classifier_tree, test_set2))
print("devtest_set: ", nltk.classify.accuracy(classifier_tree, devtest_set2))
```

```
train_set:  0.6635017701963308
test_set:  0.66
devtest_set:  0.662
```

Very surprisingly, despite that this is a very simple feature, using the first letter along is quite telling of a persons' gender. Even that this is a very simple feature, it alone reaches the accuracy level of our first feature (very extensive feature)

Feature - Last 2 Letters

In [32]:
```
## FEATURE - LAST TWO LETTERS
def last_2_letters(name): ###
    name = name.lower()
    return {
        'last_2_letters': name[-2:]
        }
last_2_letters("Andy")
```

Out[32]:
```
{'last_2_letters': 'dy'}
```

In [33]:
```
train_set3 = [(last_2_letters(n), g) for (n,g) in train_set] ###
devtest_set3 = [(last_2_letters(n), g) for (n,g) in devtest_set] ###
test_set3 = [(last_2_letters(n), g) for (n,g) in test_set] ###

nb3 = nltk.NaiveBayesClassifier.train(train_set3)  ###

print('Devset accuracy is')
print(nltk.classify.accuracy(nb3, devtest_set3)) ###
print('Test accuracy is')
print(nltk.classify.accuracy(nb3, test_set3)) ###
```

```
Devset accuracy is
0.84
Test accuracy is
0.83
```

In [34]:
```
performance_metrics(nb3, train_set3) ###
```

```
Male precision:  0.8169
Female precision:  0.8366
Male recall:  0.6796
Female recall:  0.915
```

In [35]:
```
performance_metrics(nb3, devtest_set3) ###
```

```
Male precision:  0.8039
Female precision:  0.8559
Male recall:  0.711
Female recall:  0.9083
```

In [36]:
```
# Looking at the most informative features
nb3.show_most_informative_features(10)
```

```
Most Informative Features
          last_2_letters = 'na'           female : male   =    148.5 : 1.0
          last_2_letters = 'la'           female : male   =     66.1 : 1.0
          last_2_letters = 'us'             male : female =     57.7 : 1.0
          last_2_letters = 'ra'           female : male   =     51.7 : 1.0
          last_2_letters = 'ia'           female : male   =     49.3 : 1.0
          last_2_letters = 'rt'             male : female =     31.8 : 1.0
          last_2_letters = 'do'             male : female =     26.2 : 1.0
          last_2_letters = 'rd'             male : female =     26.2 : 1.0
          last_2_letters = 'ld'             male : female =     22.7 : 1.0
          last_2_letters = 'ch'             male : female =     21.6 : 1.0
```

As expected, using the last 2 letters, the performance is very satisfying.

And very interestingly, the last 2 letters have a very good recall imale name, strikingly different from the first letter.

Here, the mail Rico is 0.70, which means only 30% of false positive rate, much better than the first letter.

This feature have a similar performance all female, compared to first letter

Feature 4- Combo of Letters Vowels

Next we want to utilize the vector training method that is in the textbook.

We want to see if this vectorizing training will help or not to help in classifying gender by names

In [ ]:
```

```

In [ ]:
```

```

In [38]:
```
def gender_features2nd(name):
    name = name.lower()
    return{
        'first_letter': name[0].lower(),
        'first2_letter': name[0:2].lower(),
        'first3_letter': name[0:3].lower(),
        'last_letter': name[-1].lower(),
        'last2_letter': name[-2:].lower(),
        'last3_letter': name[-3:].lower(),
```

```
        'last_vowel': (name[-1] in 'aeiou')
    }
```

In [39]:
```
gender_features2nd("Mary")
```

Out[39]:
```
{'first_letter': 'm',
 'first2_letter': 'ma',
 'first3_letter': 'mar',
 'last_letter': 'y',
 'last2_letter': 'ry',
 'last3_letter': 'ary',
 'last_vowel': False}
```

Vectorize the Features (Feature)

In [40]:
```
import numpy as np

# Vectorize the features function
features = np.vectorize(gender_features2nd)

print(features(['Mary']))
```

```
[{'first_letter': 'm', 'first2_letter': 'ma', 'first3_letter': 'mar', 'last_letter':
'y', 'last2_letter': 'ry', 'last3_letter': 'ary', 'last_vowel': False}]
```

In [41]:
```
# Extract the features for entire dataset
X = np.array(features(names))[:, 0] # X contains the features

# Get the gender column
y = np.array(names)[:, 1]           # y contains the targets

print("Name: %s, features=%s, gender=%s" % (names[0][0], X[0], y[0]))
```

```
Name: Idaline, features={'first_letter': 'i', 'first2_letter': 'id', 'first3_letter': 'i
da', 'last_letter': 'e', 'last2_letter': 'ne', 'last3_letter': 'ine', 'last_vowel': Tru
e}, gender=female
```

In [42]:
```
# Shuffle and split:  train, dev-test, test
from sklearn.utils import shuffle
X,y = shuffle(X,y)

X_test, X_dev_test, X_train = X[:500], X[500:1000], X[1000:]
y_test, y_dev_test, y_train = y[:500], y[500:1000], y[1000:]

print("test: " , len(X_test))
print("devtest: ", len(X_dev_test))
print("train: ", len(X_train))
```

```
test:  500
devtest:  500
train:  6214
```

Use vectorizer to transform the features into feature-vectors.

In [43]:
```
from sklearn.feature_extraction import DictVectorizer

print(features(['Mary', 'Joe']))

print(features(['Mary']))
```

```
print(features(['Joe']))

# train the vectorizer to know the possible features and values.
vectorizer = DictVectorizer()
vectorizer.fit(X_train)

transform = vectorizer.transform(features(['Mary', 'Joe']))
print(transform)
print(type(transform))
print(transform.toarray()[0][12])
print(vectorizer.feature_names_[12])
```

```
[{'first_letter': 'm', 'first2_letter': 'ma', 'first3_letter': 'mar', 'last_letter':
'y', 'last2_letter': 'ry', 'last3_letter': 'ary', 'last_vowel': False}
 {'first_letter': 'j', 'first2_letter': 'jo', 'first3_letter': 'joe', 'last_letter':
'e', 'last2_letter': 'oe', 'last3_letter': 'joe', 'last_vowel': True}]
[{'first_letter': 'm', 'first2_letter': 'ma', 'first3_letter': 'mar', 'last_letter':
'y', 'last2_letter': 'ry', 'last3_letter': 'ary', 'last_vowel': False}]
[{'first_letter': 'j', 'first2_letter': 'jo', 'first3_letter': 'joe', 'last_letter':
'e', 'last2_letter': 'oe', 'last3_letter': 'joe', 'last_vowel': True}]
  (0, 127)      1.0
  (0, 969)      1.0
  (0, 1504)     1.0
  (0, 1716)     1.0
  (0, 1855)     1.0
  (0, 3085)     1.0
  (0, 3087)     0.0
  (1, 108)      1.0
  (1, 829)      1.0
  (1, 1501)     1.0
  (1, 1674)     1.0
  (1, 2319)     1.0
  (1, 3066)     1.0
  (1, 3087)     1.0
<class 'scipy.sparse.csr.csr_matrix'>
0.0
first2_letter=ap
```

Classifier - DecisionTree¶

On Vectorized Transformed Data

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
# DT classifier to extract discriminating rules from the features.
DT_classifier = DecisionTreeClassifier()

DT_classifier.fit(vectorizer.transform(X_train), y_train)
```

Out[44]:
```
DecisionTreeClassifier()
```

In [45]:
```
print(DT_classifier.predict(vectorizer.transform(features(["Sebastian", "Amy"]))))
```

```
['male' 'female']
```

In [46]:
```
# Accuracy
print("Accuracy on training set: ", DT_classifier.score(vectorizer.transform(X_train),
print("Accuracy on dev-test set: ",DT_classifier.score(vectorizer.transform(X_dev_test)
print("Accuracy on test set: ",DT_classifier.score(vectorizer.transform(X_test), y_test
```

```
Accuracy on training set:  0.9988735114258127
Accuracy on dev-test set:  0.808
```

```
Accuracy on test set:  0.854
```

Cross Validation

In [47]:
```
# cross validation
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score

pred_train = cross_val_predict(DT_classifier, vectorizer.transform(X_train), y_train, c
pred_dev_test = cross_val_predict(DT_classifier, vectorizer.transform(X_dev_test), y_de
pred_test = cross_val_predict(DT_classifier, vectorizer.transform(X_test), y_test, cv =

score_train = accuracy_score(y_train, pred_train)
score_dev_test = accuracy_score(y_dev_test, pred_dev_test)
score_test = accuracy_score(y_test, pred_test)

print("Cross Validation")
print("Train Score = {0:5f}".format(score_train))
print("Dev Test Score = {0:5f}".format(score_dev_test))
print("Test Score = {0:5f}".format(score_test))
```

```
Cross Validation
Train Score = 0.853235
Dev Test Score = 0.800000
Test Score = 0.774000
```

Summary

We have found that The simple feature, such as first letter alone, last 2 letters, perform almost equally well as most extensive features that take into consideration One letter, 2 letters, first letters, last letters, prefix, suffix, and the vowel

However, certain features (last two letters) perform well on female, Perform very poorly on mmale in recall.

Baysians classification and is ishant re perfo....ly well in the features that we chose

The validation of development test set are satisfied cimit

In [ ]:
```
#https://www.nltk.org/book/ch06.html
```