In [2]:

```
import sys
import nltk
import numpy as np
import pandas as pd
import sklearn

print('Python:{}'.format(sys.version))
print('Nltk:{}'.format(nltk.__version__))
print('Numpy:{}'.format(np.__version__))
print('Pandas:{}'.format(pd.__version__))
print('Sklearn:{}'.format(sklearn.__version__))
```

```
Python:3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Nltk:3.4.5
Numpy:1.18.1
Pandas:0.24.2
Sklearn:0.22.1
```

In [10]:

```
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/inde
x.xml (https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml)
```

Out[10]:

```
True
```

Corpus -

A collection of written texts, especially the entire works of a particular author or a body of writing on a particular subject. Corpora is the plural of this. Example: A collection of medical journals.

Lexicon -

the vocabulary of a person, language, or branch of knowledge. Example: English dictionary. Consider, however, that various fields will have different lexicons.

Token -

Each "entity" that is a part of whatever was split up based on rules. For examples, each word is a token when a sentence is "tokenized" into words. Each sentence can also be a token, if you tokenized the sentences out of a paragraph.

In [3]:

```
from nltk.tokenize import sent_tokenize, word_tokenize

text = 'This is first sentence. Here comes second sentence, you sure? This one is last sent
```

In [4]:

```
print(sent_tokenize(text))
print(word_tokenize(text))
```

```
['This is first sentence.', 'Here comes second sentence, you sure?', 'This o
ne is last sentence.']
['This', 'is', 'first', 'sentence', '.', 'Here', 'comes', 'second', 'sentenc
e', ',', 'you', 'sure', '?', 'This', 'one', 'is', 'last', 'sentence', '.']
```

Stop Words with NLTK:

When using Natural Language Processing, our goal is to perform some analysis or processing so that a computer can respond to text appropriately.

The process of converting data to something a computer can understand is referred to as "pre-processing." One of the major forms of pre-processing is going to be filtering out useless data. In natural language processing, useless words (data), are referred to as stop words.

In [5]:

```
from nltk.corpus import stopwords
print(set(stopwords.words('english')))
```

```
{'yourselves', 'herself', 'yourself', 'of', 'am', 'again', 'these', 'did',
"weren't", 'having', 'ours', 'and', 'you', 'below', 'all', 'during', 'that',
'mustn', 'd', 'when', 'by', 'before', "hadn't", 'ourselves', 'here', 'do',
'about', 'very', 'which', 'both', 'the', 'once', 'aren', 'through', 'my', 't
hen', 'couldn', 'where', 'hadn', 'for', 'from', 'isn', 'down', 'doing', 'whi
le', "should've", 'had', "shan't", 'they', 'an', 've', 'him', 'why', 'whom',
'as', "wouldn't", "you're", 'such', 'can', 'were', "you'd", 'no', 'won', 'fe
w', 'hers', "you'll", 'until', 'her', 'under', 'what', "hasn't", 'not', "is
n't", 'up', "won't", 'on', 'those', 'further', 'your', 'themselves', 'some',
'it', 'himself', 'them', 'weren', 'are', "didn't", 'who', 'she', "haven't",
'a', 'into', 'so', 'haven', "needn't", 'than', "that'll", 'or', 'yours', 'wi
th', 'ma', 'after', 'don', 'his', 'was', 'our', "she's", 'there', "should
n't", 'at', 'wouldn', 'ain', 'just', 'off', 'against', 'this', 'shan', 'i',
'too', 'other', 'same', 'wasn', 'itself', 'has', 'now', 'any', 'o', 't', 'm
e', 'over', 're', 'but', 'y', 'needn', 'didn', 'how', "mustn't", "aren't",
'above', 's', 'mightn', "don't", 'myself', 'm', 'll', 'have', 'each', 'in',
'nor', 'most', 'does', "mightn't", 'should', "you've", 'been', 'be', "could
n't", 'to', 'being', 'doesn', 'only', 'will', 'is', 'because', 'hasn', "i
t's", "wasn't", 'we', 'between', 'shouldn', 'he', 'their', 'if', 'own', "doe
sn't", 'out', 'theirs', 'more', 'its'}
```

In [6]:

```
example = 'This is just an example to filter out stop words.'
stop_words =set(stopwords.words('english'))
word_tokens = word_tokenize(example)
filtered_words = [w for w in word_tokens if w not in stop_words]
print(word_tokens)
print(filtered_words)
```

```
['This', 'is', 'just', 'an', 'example', 'to', 'filter', 'out', 'stop', 'word
s', '.']
['This', 'example', 'filter', 'stop', 'words', '.']
```

Stemming Words with NLTK:

Stemming, which attempts to normalize sentences, is another preprocessing step that we can perform. In the english language, different variations of words and sentences often having the same meaning. Stemming is a way to account for these variations; furthermore, it will help us shorten the sentences and shorten our lookup. For example, consider the following sentence:

- I was taking a ride on my horse.
- I was riding my horse.

These sentences mean the same thing, as noted by the same tense (-ing) in each sentence; however, that isn't intuitively understood by the computer. To account for all the variations of words in the english language, we can use the Porter stemmer, which has been around since 1979.

In [7]:

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

example_words = ['coding', 'coder', 'code', 'codes']
done = [ps.stem(w) for w in example_words]
print(done)
```

```
['code', 'coder', 'code', 'code']
```

In [8]:

```
# Stemming entire sentence

new_text = "When riders are riding their bikes, they often think of racers race on th racet
words=word_tokenize(new_text)

print([ps.stem(w) for w in words])
```

```
['when', 'rider', 'are', 'ride', 'their', 'bike', ',', 'they', 'often', 'thi
nk', 'of', 'racer', 'race', 'on', 'th', 'racetrack', 'with', 'their', 'supe
r', 'bike', '.']
```

Part of Speech Tagging with NLTK

Part of speech tagging means labeling words as nouns, verbs, adjectives, etc. Even better, NLTK can handle tenses! While we're at it, we are also going to import a new sentence tokenizer (PunktSentenceTokenizer). This tokenizer is capable of unsupervised learning, so it can be trained on any body of text.

In [9]:

```
from nltk.corpus import udhr
print(udhr.raw('English-Latin1'))
```

```
Everyone has the right to freedom of thought, conscience and religion; thi
s right includes freedom to change his religion or belief, and freedom, ei
ther alone or in community with others and in public or private, to manife
st his religion or belief in teaching, practice, worship and observance.

Article 19
Everyone has the right to freedom of opinion and expression; this right in
cludes freedom to hold opinions without interference and to seek, receive
and impart information and ideas through any media and regardless of front
iers.

Article 20
Everyone has the right to freedom of peaceful assembly and association.
No one may be compelled to belong to an association.
Article 21
Everyone has the right to take part in the government of his country, dire
ctly or through freely chosen representatives.
Everyone has the right to equal access to public service in his country.
The will of the people shall be the basis of the authority of government;
this will shall be expressed in periodic and genuine elections which shall
```

In [10]:

```
# Lets import some sample and training text - George Bush's 2005 and 2006 state of the unio

from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer

train_text = state_union.raw("2005-GWBush.txt")
sample_text = state_union.raw("2006-GWBush.txt")
```

In [11]:

```
# Train the PunktSentenceTokenizer

custom_sent_tokenizer = PunktSentenceTokenizer(train_text)
```

In [12]:

```
# Tokenize the sample_text using our trained tokenizer

tokenized = custom_sent_tokenizer.tokenize(sample_text)
```

In [13]:

```
# Function will tag each tokenized word with a part of speech

def language_process():
    try:
        for i in tokenized[:5]:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            print(tagged)

    except Exception as e:
        print(str(e))

language_process()
```

```
[('PRESIDENT', 'NNP'), ('GEORGE', 'NNP'), ('W.', 'NNP'), ('BUSH', 'NNP'),
("'S", 'POS'), ('ADDRESS', 'NNP'), ('BEFORE', 'IN'), ('A', 'NNP'), ('JOINT',
'NNP'), ('SESSION', 'NNP'), ('OF', 'IN'), ('THE', 'NNP'), ('CONGRESS', 'NN
P'), ('ON', 'NNP'), ('THE', 'NNP'), ('STATE', 'NNP'), ('OF', 'IN'), ('THE',
'NNP'), ('UNION', 'NNP'), ('January', 'NNP'), ('31', 'CD'), (',', ','), ('20
06', 'CD'), ('THE', 'NNP'), ('PRESIDENT', 'NNP'), (':', ':'), ('Thank', 'NN
P'), ('you', 'PRP'), ('all', 'DT'), ('.', '.')]
[('Mr.', 'NNP'), ('Speaker', 'NNP'), (',', ','), ('Vice', 'NNP'), ('Presiden
t', 'NNP'), ('Cheney', 'NNP'), (',', ','), ('members', 'NNS'), ('of', 'IN'),
('Congress', 'NNP'), (',', ','), ('members', 'NNS'), ('of', 'IN'), ('the',
'DT'), ('Supreme', 'NNP'), ('Court', 'NNP'), ('and', 'CC'), ('diplomatic',
'JJ'), ('corps', 'NN'), (',', ','), ('distinguished', 'JJ'), ('guests', 'NN
S'), (',', ','), ('and', 'CC'), ('fellow', 'JJ'), ('citizens', 'NNS'), (':',
':'), ('Today', 'VB'), ('our', 'PRP$'), ('nation', 'NN'), ('lost', 'VBD'),
('a', 'DT'), ('beloved', 'VBN'), (',', ','), ('graceful', 'JJ'), (',', ','),
('courageous', 'JJ'), ('woman', 'NN'), ('who', 'WP'), ('called', 'VBD'), ('A
merica', 'NNP'), ('to', 'TO'), ('its', 'PRP$'), ('founding', 'NN'), ('ideal
s', 'NNS'), ('and', 'CC'), ('carried', 'VBD'), ('on', 'IN'), ('a', 'DT'),
('noble', 'JJ'), ('dream', 'NN'), ('.', '.')]
[('Tonight', 'NN'), ('we', 'PRP'), ('are', 'VBP'), ('comforted', 'VBN'), ('b
y', 'IN'), ('the', 'DT'), ('hope', 'NN'), ('of', 'IN'), ('a', 'DT'), ('gla
d', 'JJ'), ('reunion', 'NN'), ('with', 'IN'), ('the', 'DT'), ('husband', 'N
N'), ('who', 'WP'), ('was', 'VBD'), ('taken', 'VBN'), ('so', 'RB'), ('long',
'RB'), ('ago', 'RB'), (',', ','), ('and', 'CC'), ('we', 'PRP'), ('are', 'VB
P'), ('grateful', 'JJ'), ('for', 'IN'), ('the', 'DT'), ('good', 'JJ'), ('lif
e', 'NN'), ('of', 'IN'), ('Coretta', 'NNP'), ('Scott', 'NNP'), ('King', 'NN
P'), ('.', '.')]
[('(', '('), ('Applause', 'NNP'), ('.', '.'), (')', ')')]
[('President', 'NNP'), ('George', 'NNP'), ('W.', 'NNP'), ('Bush', 'NNP'),
('reacts', 'VBZ'), ('to', 'TO'), ('applause', 'VB'), ('during', 'IN'), ('hi
s', 'PRP$'), ('State', 'NNP'), ('of', 'IN'), ('the', 'DT'), ('Union', 'NN
P'), ('Address', 'NNP'), ('at', 'IN'), ('the', 'DT'), ('Capitol', 'NNP'),
(',', ','), ('Tuesday', 'NNP'), (',', ','), ('Jan', 'NNP'), ('.', '.')]
```

In [14]:

```
import nltk
nltk.download('tagsets')
nltk.help.upenn_tagset()
```

```
JJS: adjective, superlative
    calmest cheapest choicest classiest cleanest clearest closest commones
t
    corniest costliest crassest creepiest crudest cutest darkest deadliest
    dearest deepest densest dinkiest ...
LS: list item marker
    A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005
    SP-44007 Second Third Three Two * a b c d first five four one six thre
e
    two
MD: modal auxiliary
    can cannot could couldn't dare may might must need ought shall should
    shouldn't will would
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino afghan shed thermostat
    investment slide humour falloff slick wind hyena override subhumanity
    machinist ...
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos
    Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA
    Shannon A.K.C. Meltex Liverpool
```

Chunking with NLTK

Now that each word has been tagged with a part of speech, we can move onto chunking: grouping the words into meaningful clusters. The main goal of chunking is to group words into "noun phrases", which is a noun with any associated verbs, adjectives, or adverbs.

The part of speech tags that were generated in the previous step will be combined with regular expressions, such as the following:

- ' + ' = match 1 or more
- ' ? ' = match 0 or 1 repetitions.
- ' * ' = match 0 or MORE repetitions
- ' . ' = Any character except a new line

In [15]:

```
train_text = state_union.raw("2005-GWBush.txt")
sample_text = state_union.raw("2006-GWBush.txt")

custom_sent_tokenizer = PunktSentenceTokenizer(train_text)

tokenized = custom_sent_tokenizer.tokenize(sample_text)

def language_process():
    try:
        for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)

            # combine the part-of-speech tag with a regular expression

            chunkGram = r"""Chunk: {<RB.?>*<VB.?>*<NNP>+<NN>?}"""
            chunkParser = nltk.RegexpParser(chunkGram)
            chunked = chunkParser.parse(tagged)

            # draw the chunks with nltk
            #chunked.draw()

    except Exception as e:
        print(str(e))


language_process()
```

In [ ]:

```
The main line in question is:
'''
chunkGram = r"""Chunk: {<RB.?>*<VB.?>*<NNP>+<NN>?}"""
'''
This line, broken down:
'''
<RB.?>* = "0 or more of any tense of adverb," followed by:

<VB.?>* = "0 or more of any tense of verb," followed by:

<NNP>+ = "One or more proper nouns," followed by

<NN>? = "zero or one singular noun."

'''
```

In [16]:

```
def language_process():
    try:
        for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)

            # combine the part-of-speech tag with a regular expression

            chunkGram = r"""Chunk: {<RB.?>*<VB.?>*<NNP>+<NN>?}"""
            chunkParser = nltk.RegexpParser(chunkGram)
            chunked = chunkParser.parse(tagged)

            # print(chunked)
            for subtree in chunked.subtrees(filter=lambda t: t.label() == 'Chunk'):
                print(subtree)

            # draw the chunks with nltk
            # chunked.draw()

    except Exception as e:
        print(str(e))


language_process()
```

```
(Chunk PRESIDENT/NNP GEORGE/NNP W./NNP BUSH/NNP)
(Chunk ADDRESS/NNP)
(Chunk A/NNP JOINT/NNP SESSION/NNP)
(Chunk THE/NNP CONGRESS/NNP ON/NNP THE/NNP STATE/NNP)
(Chunk THE/NNP UNION/NNP January/NNP)
(Chunk THE/NNP PRESIDENT/NNP)
(Chunk Thank/NNP)
(Chunk Mr./NNP Speaker/NNP)
(Chunk Vice/NNP President/NNP Cheney/NNP)
(Chunk Congress/NNP)
(Chunk Supreme/NNP Court/NNP)
(Chunk called/VBD America/NNP)
(Chunk Coretta/NNP Scott/NNP King/NNP)
(Chunk Applause/NNP)
(Chunk President/NNP George/NNP W./NNP Bush/NNP)
(Chunk State/NNP)
(Chunk Union/NNP Address/NNP)
(Chunk Capitol/NNP)
(Chunk Tuesday/NNP)
```

Chinking with NLTK

Sometimes there are words in the chunks that we don't won't, we can remove them using a process called chinking.

In [17]:

```
def language_process():
    try:
        for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)

            # combine the part-of-speech tag with a regular expression

            chunkGram = r"""Chunk: {<.*>+}
                                }<VB.?|IN|DT|TO>+{"""
            chunkParser = nltk.RegexpParser(chunkGram)
            chunked = chunkParser.parse(tagged)

            # print(chunked)
            for subtree in chunked.subtrees(filter=lambda t: t.label() == 'Chunk'):
                print(subtree)

            # draw the chunks with nltk
            # chunked.draw()

    except Exception as e:
        print(str(e))


language_process()
```

```
(Chunk Coretta/NNP Scott/NNP King/NNP ./.)
(Chunk (/( Applause/NNP ./. )/))
(Chunk President/NNP George/NNP W./NNP Bush/NNP)
(Chunk his/PRP$ State/NNP)
(Chunk Union/NNP Address/NNP)
(Chunk Capitol/NNP ,/, Tuesday/NNP ,/, Jan/NNP ./.)
(Chunk 31/CD ,/, 2006/CD ./.)
(Chunk White/NNP House/NNP photo/NN)
(Chunk Eric/NNP DraperEvery/NNP time/NN I/PRP)
(Chunk invited/JJ)
(Chunk rostrum/NN ,/, I/PRP)
(Chunk privilege/NN ,/, and/CC mindful/NN)
(Chunk history/NN we/PRP)
(Chunk together/RB ./.)
(Chunk We/PRP)
(Chunk Capitol/NNP dome/NN)
(Chunk moments/NNS)
(Chunk national/JJ mourning/NN and/CC national/JJ achievement/NN ./.)
(Chunk We/PRP)
(Chunk America/NNP)
```

Named Entity Recognition with NLTK

One of the most common forms of chunking in natural language processing is called "Named Entity Recognition." NLTK is able to identify people, places, things, locations, monetary figures, and more.

There are two major options with NLTK's named entity recognition: either recognize all named entities, or recognize named entities as their respective type, like people, places, locations, etc.

Here, with the option of binary = True, this means either something is a named entity, or not. There will be no further detail.

In [21]:

```
def language_process():
    try:
        for i in tokenized[:5]:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            namedEnt = nltk.ne_chunk(tagged, binary=True)
            namedEnt.draw()

    except Exception as e:
        print(str(e))


language_process()
```

In [20]:

```
def language_process():
    try:
        for i in tokenized[:5]:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            namedEnt = nltk.ne_chunk(tagged, binary=False)
            namedEnt.draw()

    except Exception as e:
        print(str(e))


language_process()
```

Text Classification

Text classification using NLTK

Now that we have covered the basics of preprocessing for Natural Language Processing, we can move on to text classification using simple machine learning classification algorithms.

In [29]:

```
import random
import nltk
from nltk.corpus import movie_reviews

documents = [(list(movie_reviews.words(fileid)),category)
             for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]

# shuffle the documents
random.shuffle(documents)

print('Number of Documents: {}'.format(len(documents)))
print('First Review: {}'.format(documents[1]))

all_words = []
for w in movie_reviews.words():
    all_words.append(w.lower())

all_words = nltk.FreqDist(all_words)

print('')
print('Most common words: {}'.format(all_words.most_common(15)))
print('The word happy: {}'.format(all_words["happy"]))
```

```
Number of Documents: 2000
First Review: (['i', 'can', 'hear', 'the', 'question', 'already', '.', 'wh
at', 'on', 'earth', 'do', 'these', 'two', 'movies', 'have', 'in', 'commo
n', '?', 'to', 'most', 'people', ',', 'not', 'a', 'lot', ',', 'except', 't
hat', 'both', 'are', 'by', 'renowned', 'directors', '.', 'as', 'i', 'saw',
'them', ',', 'however', ',', 'both', 'movies', 'have', 'flawed', 'romanti
c', 'scripts', 'wrapped', 'in', 'distinctive', 'packaging', 'of', 'lavis
h', 'visuals', 'musical', 'numbers', '.', 'but', 'oh', ',', 'how', 'differ
ently', 'the', 'packages', 'affect', 'their', 'films', '.', 'while', '"',
'everyone', "'", 's', '"', 'production', 'numbers', 'make', 'an', 'otherwi
se', 'ordinary', 'woody', 'tale', 'something', 'special', ',', 'jane', 'ca
mpion', "'", 's', 'imaginative', 'visuals', 'only', 'serve', 'to', 'emphas
ize', 'how', 'pompous', 'and', 'uninvolving', 'laura', 'jones', "'", 'scri
pt', 'is', '.', 'i', 'left', '"', 'everyone', 'says', 'i', 'love', 'you',
'"', 'not', 'remembering', 'a', 'lot', 'about', 'who', 'loved', 'whom',
',', 'but', 'its', 'infectious', 'happiness', 'put', 'a', 'grin', 'on', 'm
y', 'face', '.', 'i', 'left', '"', 'the', 'portrait', 'of', 'a', 'lady',
'"', 'not', 'remembering', 'a', 'lot', 'about', 'who', 'loved', 'whom',
',', 'and', 'i', 'could', 'have', 'cared', 'less', '.', '"', 'everyone',
```

In [37]:

```
# We'll use the 4000 most common words as features
print(len(all_words))
word_features = list(all_words.keys())[:4000]
print(word_features[:10])
```

```
39768
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church', 'party']
```

In [40]:

```
# The find_features function will determine which of the 3000 word features are contained i
def find_features(document):
    words = set(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    #print(features)
    return features


# Lets use an example from a negative review
features = find_features(movie_reviews.words('neg/cv000_29416.txt'))
for key, value in features.items():
    if value == True:
        print (key)
```

```
someone
assuming
genre
hot
kids
also
wrapped
production
years
ago
sitting
shelves
ever
whatever
skip
where
joblo
nightmare
elm
street
```

In [41]:

```
# Now lets do it for all the documents
featuresets = [(find_features(rev), category) for (rev, category) in documents]
```

In [55]:

```
# Split the featuresets into training and testing datasets using sklearn
from sklearn import model_selection

# define a seed for reproducibility
seed = 1

# split the data into training and testing datasets
training, testing = model_selection.train_test_split(featuresets, test_size = 0.25, random_

print(len(training))
print(len(testing))
#print(training[:2])
```

```
1500
500
```

In [51]:

```
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# define scoring method
scoring = 'accuracy'

# Define models to train
models=[]

models.append(('Nearest Neighbors',SklearnClassifier(KNeighborsClassifier(n_neighbors = 2))
models.append(('Decision Tree',SklearnClassifier(DecisionTreeClassifier(max_depth=5))))
models.append(('Random Forest',SklearnClassifier(RandomForestClassifier(max_depth=5, n_esti
models.append(('Neural Net',SklearnClassifier(MLPClassifier(alpha=1, max_iter=400, warm_sta
models.append(('AdaBoost',SklearnClassifier(AdaBoostClassifier())))
models.append(('SVM Linear',SklearnClassifier(SVC(kernel = 'linear'))))
models.append(('SVM RBF',SklearnClassifier(SVC(kernel = 'rbf'))))
models.append(('SVM Sigmoid',SklearnClassifier(SVC(kernel = 'sigmoid'))))
models.append(('SVM Polynomial',SklearnClassifier(SVC(kernel = 'poly'))))

# evaluate each model in turn
results = []
names = []

for name, model in models:
    model.train(training)
    # and test on the testing dataset!
    accuracy = nltk.classify.accuracy(model, testing)*100
    print("{}: {}".format(name,accuracy))
```

```
Nearest Neighbors: 52.800000000000004
Decision Tree: 61.199999999999996
Random Forest: 52.400000000000006
Neural Net: 81.2
AdaBoost: 79.0
SVM Linear: 80.2
SVM RBF: 84.2
SVM Sigmoid: 82.39999999999999
SVM Polynomial: 84.2
```