# Howitt_NLP_FINAL_v4submit

May 19, 2016

In [1]:
```
import urllib #for pulling from urls
import bs4 #beautiful soup for cleaning html
import nltk
import pandas as pd
import sklearn
import re #regex
import glob #for importing files
from itertools import izip_longest
from nltk.tag import StanfordPOSTagger
import numpy as np
import os
from sklearn.feature_extraction.text import CountVectorizer
os.environ['JAVAHOME'] = "C:\Program Files (x86)\Java\jre1.8.0_91\\bin\java.exe"
# constants for tagger instantiation; these get kinda unweildy inline
STANFORD_TAGGER_HOME = 'C:\Python27\stanford-postagger-full-2014-08-27\stanford-postagger-full-
TAGGER_JAR = "stanford-postagger.jar"
#BD_DISTSIM_MODEL = "models\english-bidirectional-distsim.tagger"
L3_DISTSIM_MODEL = "models\english-left3words-distsim.tagger"
OPT_HEAP_SIZE = '-mx1200m'
```

In [2]:
```
# helper functions

# splits an iterable into chunks (http://stackoverflow.com/questions/434287/what-is-the-most-py
def group(group_size, iterable):
    #a little magic, but it's a list of n copies of the same iterator over the iterable
    args = [iter(iterable)] * group_size
    #makes a list that's like [args[0][0], args[1][0], args[2][0]]...kinda.
    #see https://docs.python.org/2/library/itertools.html?highlight=izip_longest#itertools.izip_
    # -- https://docs.python.org/2/library/functions.html#iter
    # -- https://docs.python.org/2/glossary.html#term-iterator
    return izip_longest(fillvalue=None, *args)

# a little functional helper for filtering things
def isTrue(x):
    return True if x else False
```

In [3]:
```
def openTXT(filename):
    article = open(filename, "r")
    articletext = article.read()
    #opentext takes a file and reads it
    return articletext


occupyemergent = [openTXT(text) for text in (glob.glob("C:\Python27\occupyemergent*.txt"))]
```

```
#list comprehension opens a file and reads it and stores it in a list
privilegeemergent =[openTXT(text) for text in (glob.glob("C:\Python27\privilegeemergent*.txt"))]

occupystandard = [openTXT(text) for text in (glob.glob("C:\Python27\occupystandard*.txt"))]
privilegestandard = [openTXT(text) for text in (glob.glob("C:\Python27\privilegestandard*.txt"))]

occupytrain = occupyemergent[:15] + occupystandard[:15]
occupytest = occupyemergent[15:] + occupystandard[15:]
#training data contains 30 instances, test data contains 10
privilegetrain = privilegeemergent[:15] + privilegestandard[:15]
privilegetest = privilegeemergent[15:] + privilegestandard[15:]


#print len(occupyemergent)
#print len(occupystandard)
#print len(occupytrain)
#print len(occupytest)
#print len(privilegeemergent)
#print len(privilegestandard)
#print len(privilegetrain)
#print len(privilegetest)

occupytrainclean = [text.decode('ascii', errors = 'ignore') for text in occupytrain]
occupytestclean = [text.decode('ascii', errors = 'ignore') for text in occupytest]
#above code removes ascii from files
privilegetrainclean = [text.decode('ascii', errors = 'ignore') for text in privilegetrain]
privilegetestclean = [text.decode('ascii', errors = 'ignore') for text in privilegetest]

#print (glob.glob("C:\Python27\occupyemergent*.txt"))

#print occupytrainclean[7]
```

In [5]:
```
# takes one article and returns a list of (word, tag)
def POStag(cleanedtext):
    taggedcleantext = []
    #tagged = st.tag(cleanedtext.split())
    #taggedcleantext.append(tagged)
    #giving smaller chunks to the tagger to avoid OutOfMemory issues
    sents = cleanedtext.split(".")
    for chunk in group(50, sents):
        st = StanfordPOSTagger(STANFORD_TAGGER_HOME + L3_DISTSIM_MODEL, STANFORD_TAGGER_HOME +
        chunk = filter(isTrue, chunk) # filter out the None's that 'group' leaves in the final
        chunk = ".".join(chunk) # regroup chunks into strings instead of lists of sentences
        chunk = chunk.split()
        taggedcleantext.extend(st.tag(chunk))

    return taggedcleantext

def runPOS(listofarticles):
    taggedarticles = []
    for article in listofarticles:
        taggedarticle = POStag(article)
        taggedarticles.append(taggedarticle)
    return taggedarticles
```

```
        taggedoccupytrain = runPOS(occupytrainclean)
        taggedoccupytest = runPOS(occupytestclean)
        taggedprivilegetrain = runPOS(privilegetrainclean)
        taggedprivilegetest = runPOS(privilegetestclean)
        #print taggedoccupytrain
```

In [8]:
```
def bagofWords(cleanedtrained, cleanedtest):
    vectorizer = CountVectorizer(min_df=1)
    X = vectorizer.fit_transform(cleanedtrained)
    X2 = vectorizer.transform(cleanedtest)
    X.toarray()

    analyze = vectorizer.build_analyzer()
    return X.toarray(), X2.toarray()


occbagtrain, occupybagtest = bagofWords(occupytrainclean, occupytestclean)
privbagtrain, privbagtest = bagofWords(privilegetrainclean, privilegetestclean)

print len(occbagtrain[0]), len(occupybagtest[0])
```

```
6186 6186
```

In [4]:
```
occupyregex = r"\b[Oo]ccup(y|ation|ies)\b"
occupyregexcap = r"\bOccup(y|ation|ies)\b"

privilegeregex = r"\b[Pp]rivileges?\b"
privilegeregexcap = r"\bPrivileges?\b"

phobicregex = r"\b((\w)+)?[Pp]hobi(c|a)s?\b"
phobicregexcap = r"\bPhobi(c|a)s?\b"

def numOccur(article, regex):
    occurlist = re.findall(regex, article)
    occurences = len(occurlist)

    return occurences

#print numOccur(occupytrainclean[0], occupyregex)
#print numOccur(privilegetrainclean[0], privilegeregex)

def findCap(article, regex):
    caplist = re.findall(regex, article)
    capoccurence = len(caplist)

    return capoccurence

#print findCap(articletext, occupyregexcap)

def runTime(listofarticles, regex, regexcap):
    occurencelist = []
    capslist = []
    lenoart = []
    for article in listofarticles:
```

```
            numo = numOccur(article, regex)
            occurencelist.append(numo)
            caps = findCap(article, regexcap)
            capslist.append(caps)
            leng = len(article)
            lenoart.append(leng)
        return occurencelist, capslist, lenoart

    #otc = [x for i,x in enumerate(occupytrainclean) if i != 19]
    #otc19 = occupytrainclean[19]
    #otc19_sub = otc19[:500]
    #print otc19_sub

    #otc_test = occupytrainclean
    #otc_test.pop(19)
    #otc_test.append(otc19_sub)
    #print len(otc_test)

    occursprtr, capsprtr, lengthsprtr = runTime(privilegetrainclean, privilegeregex, privilegeregexc
    occursprte, capsprte, lengthsprte = runTime(privilegetestclean, privilegeregex, privilegeregexca
    occursoctr, capsoctr, lengthsoctr = runTime(occupytrainclean, occupyregex, occupyregexcap)
    occursocte, capsocte, lengthsocte = runTime(occupytestclean, occupyregex, occupyregexcap)



    #print occurs
    #print caps
    #print lengths
    #print occupytrainclean[19]
```

In [15]:
```
def removepunc(x):
    punc = ".,?!:;\\/<>@#$%^&*()[]{}\""
    return not x in punc

#filter takes function (but no arguments: so not to evaluate)
#cool
def removePuncf(text):
    newtext = filter(removepunc, text)
    return newtext

#nopunctext = removePuncf(articletext)

#print nopunctext
```

In [94]:
```
def bigrams(text, regex):
    bigram_vectorizer = CountVectorizer(ngram_range=(2, 2), token_pattern=r'\b\w+\b', min_df=1)
    analyze = bigram_vectorizer.build_analyzer()
    x2 = analyze(text)
    bigram = [gram for gram in x2  if re.findall(regex, gram)]
    return bigram
```

```
        print bigrams(privilegetrainclean[1], privilegeregex)
```

```
[u'bought privilege', u'privilege and']
```

```
In [9]: def listofPOS(text, regex):
            posstrlist = []
            for article in text:
                tupstr = ""
                for tupl in article:
                    word = tupl[0]
                    matched = re.match(regex, word)
                    if matched:
                        #print tupl
                        tupstr += tupl[1]+" "
                posstrlist.append(tupstr)
            #print posstrlist
            return posstrlist


        occupytrainPOS = listofPOS(taggedoccupytrain, occupyregex)
        occupytestPOS = listofPOS(taggedoccupytest, occupyregex)
        privilegetrainPOS = listofPOS(taggedprivilegetrain, privilegeregex)
        privilegetestPOS = listofPOS(taggedprivilegetrain, privilegeregex)



        occbagPOStrain, occbagPOStest = bagofWords(occupytrainPOS, occupytestPOS)
        privbagPOStrain, privbagPOStest = bagofWords(privilegetrainPOS, privilegetestPOS)


        print occupytrainPOS[0]
```

```
NNP VBP NNP NNP NNP NNP NNP NNP
```

```
In [16]: #set up data frame
         categories1 = ["emergent"]*15
         categories2 = ["standard"]*15
         categories3 = ["emergent"]*5
         categories4 = ["standard"]*5
         categories = categories1 + categories2
         categoriesx = categories3 + categories4

         #set up dataframe for training data
         occupy_train_data = pd.DataFrame(categories, columns=["categories"])
         occupy_train_data["occurences"] = pd.DataFrame(occursoctr)
         occupy_train_data["capitalize"] = pd.DataFrame(capsoctr)
         occupy_train_data["articlelength"] = pd.DataFrame(lengthsoctr)
         #occupy_train_data["bagofwords"] = pd.DataFrame(occbagtrain)
         #occupy_train_data["bagofPOS"] = pd.DataFrame(occbagPOStrain)



         occupy_test_data = pd.DataFrame(categoriesx, columns=["categories"])
         occupy_test_data["occurences"] = pd.DataFrame(occursocte)
         occupy_test_data["capitalize"] = pd.DataFrame(capsocte)
```

```
occupy_test_data["articlelength"] = pd.DataFrame(lengthsocte)


#print occupy_train_data
#print occupy_test_data
```

In [97]:
```
from sklearn import linear_model

X_train = occupy_train_data[occupy_train_data.columns[1:]]
y_train = occupy_train_data["categories"]

X_test = occupy_test_data[occupy_test_data.columns[1:]]
y_test = occupy_test_data["categories"]

#use logistic regression to fit and score entire training
logistic = linear_model.LogisticRegression()

logistic.fit(X_train, y_train)
logistic.score(X_test, y_test)
```

Out[97]: 1.0

In [98]:
```
from sklearn.svm import SVC

#Support Vector Classification algorithm on same data set
svc = SVC()
svc.fit(X_train, y_train)
svc.score(X_test, y_test)
```

Out[98]: 0.5

In [17]:
```
#set up dataframe for training data
privilege_train_data = pd.DataFrame(categories, columns=["categories"])
privilege_train_data["occurences"] = pd.DataFrame(occursprtr)
privilege_train_data["capitalize"] = pd.DataFrame(capsprtr)
privilege_train_data["articlelength"] = pd.DataFrame(lengthsprtr)
#privilege_train_data["bagofwords"] = pd.DataFrame(privbagtrain)
#privilege_train_data["bagofPOS"] = pd.DataFrame(privbagPOStrain)



privilege_test_data = pd.DataFrame(categoriesx, columns=["categories"])
privilege_test_data["occurences"] = pd.DataFrame(occursprte)
privilege_test_data["capitalize"] = pd.DataFrame(capsprte)
privilege_test_data["articlelength"] = pd.DataFrame(lengthsprte)

#print privilege_train_data
#print privilege_test_data
```

In [100]:
```
X_privtrain = privilege_train_data[privilege_train_data.columns[1:]]
y_privtrain = privilege_train_data["categories"]

X_privtest = privilege_test_data[privilege_test_data.columns[1:]]
y_privtest = privilege_test_data["categories"]
```

```
In [ ]: #use logistic regression to fit and score entire training
        logistic2 = linear_model.LogisticRegression()

        logistic2.fit(X_privtrain, y_privtrain)
        logistic2.score(X_privtest, y_privtest)
```

```
Out[100]: 0.59999999999999998
```

```
In [101]: svc2 = SVC()
          svc2.fit(X_privtrain, y_privtrain)
          svc2.score(X_privtest, y_privtest)
```

```
Out[101]: 0.59999999999999998
```

```
In [111]: #occupy_train_data = pd.DataFrame(categories, columms=["categories"])
          #occupy_train_data["occurences"] = pd.DataFrame(occursoctr)
          #occupy_train_data["capitalize"] = pd.DataFrame(capsoctr)
          #occupy_train_data["articlelength"] = pd.DataFrame(lengthsoctr)

          def vectorize(matrix1, matrix2, dataframe):
              labellist = []
              featurelist = []
              matrix3 = dataframe.as_matrix()
              print type(matrix1[0]), type(matrix2[0]), type(matrix3[0])
              #print int(matrix3[0][1])
              for i in range(len(dataframe)):
                  element = np.concatenate((matrix1[i], matrix2[i], matrix3[i][1:]))
                  featurelist.append(element)
                  labellist.append(matrix3[i][0])
              return featurelist, labellist

          features_train, labels_train = vectorize(occbagtrain, occbagPOStrain, occupy_train_data)

          features_test, labels_test = vectorize(occupybagtest, occbagPOStest, occupy_test_data)

          print len(features_train[0]), len(features_test[0])


          #OCCUPY SVC WITH BAG OF WORDS
          svc3 = SVC()
          svc3.fit(features_train, labels_train)
          svc3.score(features_test, labels_test)




          #print occupy_train_data.panel[0]

          #print occbagtrain[0] + occbagPOStrain[0]
```

```
<type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.ndarray'>
<type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.ndarray'>
6196 6196
```

```
Out[111]: 0.29999999999999999
```

```
In [107]: features_train2, labels_train2 = vectorize(privbagtrain, privbagPOStrain, privilege_train_data

          features_test2, labels_test2 = vectorize(privbagtest, privbagPOStest, privilege_test_data)

          #PRIVILEGE SVC WITH BOW
          svc3 = SVC()
          svc3.fit(features_train2, labels_train2)
          svc3.score(features_test2, labels_test2)

<type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.ndarray'>
<type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.ndarray'>

Out[107]: 0.59999999999999998

In [110]: #OCCUPY LOG WITH BOW

          logisticnewoc = linear_model.LogisticRegression()

          logisticnewoc.fit(features_train, labels_train)
          logisticnewoc.score(features_test, labels_test)

Out[110]: 0.90000000000000002

In [112]: #PRIVILEGE LOG WITH BOW

          logisticnewpr = linear_model.LogisticRegression()

          logisticnewpr.fit(features_train2, labels_train2)
          logisticnewpr.score(features_test2, labels_test2)

Out[112]: 0.90000000000000002

In [113]: from sklearn.naive_bayes import GaussianNB
          clf = GaussianNB()
          clf.fit(features_train, labels_train)
          clf.score(features_test, labels_test)

Out[113]: 0.59999999999999998

In [114]: clf2 = GaussianNB()
          clf2.fit(features_train2, labels_train2)
          clf2.score(features_test2, labels_test2)

Out[114]: 0.90000000000000002

In [ ]:

In [10]: phobicemergent = [openTXT(text) for text in (glob.glob("C:\Python27\phobicemergent*.txt"))]
         phobicstandard = [openTXT(text) for text in (glob.glob("C:\Python27\phobicstandard*.txt"))]

         phobictrain = phobicemergent[:15] + phobicstandard[:15]
         phobictest = phobicemergent[15:] + phobicstandard[15:]

         phobictrainclean = [text.decode('ascii', errors = 'ignore') for text in phobictrain]
         phobictestclean = [text.decode('ascii', errors = 'ignore') for text in phobictest]
         #above code removes ascii from files
```

In [118]:
```
taggedphobictrain = runPOS(phobictrainclean)
taggedphobictest = runPOS(phobictestclean)
```

In [13]:
```
#phobicbagtrain, phobicbagtest = bagofWords(phobictrainclean, phobictestclean)

occursphtr, capsphtr, lengthsphtr = runTime(phobictrainclean, phobicregex, phobicregexcap)
occursphte, capsphte, lengthsphte = runTime(phobictestclean, phobicregex, phobicregexcap)

#phobictrainPOS = listofPOS(taggedphobictrain, phobicregex)
#phobictestPOS = listofPOS(taggedphobictest, phobicregex)

#phobicbagPOStrain, phobicbagPOStest = bagofWords(phobictrainPOS, phobictestPOS)
```

In [18]:
```
#set up dataframe for training data
phobic_train_data = pd.DataFrame(categories, columns=["categories"])
phobic_train_data["occurences"] = pd.DataFrame(occursphtr)
phobic_train_data["capitalize"] = pd.DataFrame(capsphtr)
phobic_train_data["articlelength"] = pd.DataFrame(lengthsphtr)
#phobic_train_data["bagofwords"] = pd.DataFrame(phobicbagtrain)
#phobic_train_data["bagofPOS"] = pd.DataFrame(phobicbagPOStrain)



phobic_test_data = pd.DataFrame(categoriesx, columns=["categories"])
phobic_test_data["occurences"] = pd.DataFrame(occursphte)
phobic_test_data["capitalize"] = pd.DataFrame(capsphte)
phobic_test_data["articlelength"] = pd.DataFrame(lengthsphte)

#print phobic_train_data
#print phobic_test_data
```

In [123]:
```
X_train_phobic = phobic_train_data[phobic_train_data.columns[1:]]
y_train_phobic = phobic_train_data["categories"]

X_test_phobic = phobic_test_data[phobic_test_data.columns[1:]]
y_test_phobic = phobic_test_data["categories"]

#use logistic regression to fit and score entire training
logisticphob = linear_model.LogisticRegression()


#phobic log without BOW
logisticphob.fit(X_train_phobic, y_train_phobic)
logisticphob.score(X_test_phobic, y_test_phobic)
```

Out[123]:
```
0.40000000000000002
```

In [124]:
```
#phobic svc without bow
svcphob = SVC()
svcphob.fit(X_train_phobic, y_train_phobic)
svcphob.score(X_test_phobic, y_test_phobic)
```

Out[124]:
```
0.5
```

In [127]:
```
features_train3, labels_train3 = vectorize(phobicbagtrain, phobicbagPOStrain, phobic_train_da

features_test3, labels_test3 = vectorize(phobicbagtest, phobicbagPOStest, phobic_test_data)
```

```
<type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.ndarray'>
<type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.ndarray'>
```

In [128]:
```
#Phobic SVC
svcphobic = SVC()
svcphobic.fit(features_train3, labels_train3)
svcphobic.score(features_test3, labels_test3)
```

Out[128]:
```
0.59999999999999998
```

In [129]:
```
#Phobic Log
logisticnewpho = linear_model.LogisticRegression()

logisticnewpho.fit(features_train3, labels_train3)
logisticnewpho.score(features_test3, labels_test3)
```

Out[129]:
```
0.69999999999999996
```

In [130]:
```
#Phobic NB
clf3 = GaussianNB()
clf3.fit(features_train3, labels_train3)
clf3.score(features_test3, labels_test3)
```

Out[130]:
```
0.80000000000000004
```

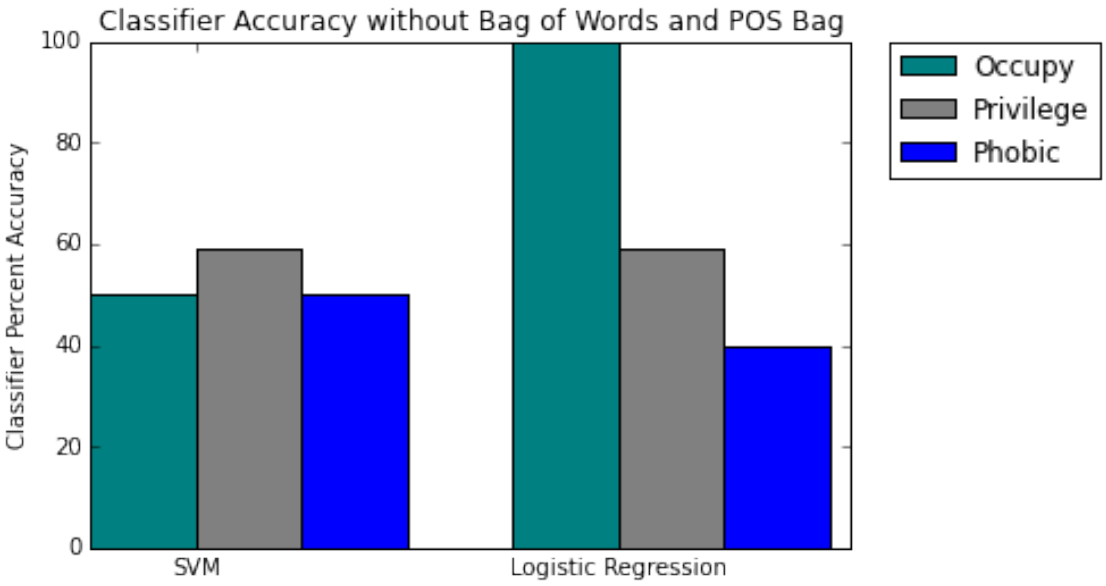
In [148]:
```
%matplotlib inline
from pylab import *

privme = [59, 59]
occme = [50, 100]
phobme = [50, 40]

placement = np.arange(2)
width = 0.25

bar(placement, occme, width, color='teal', hold=True, label = "Occupy")
bar([p+width for p in placement], privme, width, color='grey', hold=True, label = "Privilege"
bar([p+width*2 for p in placement], phobme, width, color='b', hold=True, label = "Phobic" )
ylabel("Classifier Percent Accuracy")
title("Classifier Accuracy without Bag of Words and POS Bag")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
xticks(placement + width, ('SVM', 'Logistic Regression'))
```

Out[148]:
```
([<matplotlib.axis.XTick at 0x1fc9d358>,
  <matplotlib.axis.XTick at 0x20eb8128>],
 <a list of 2 Text xticklabel objects>)
```

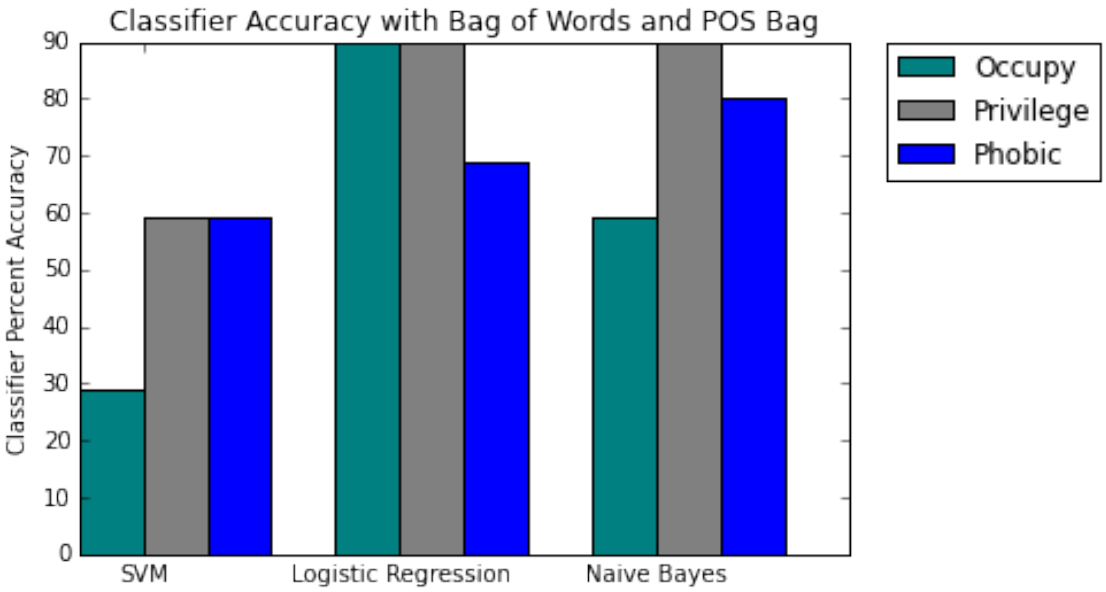



```
In [150]: privme = [59, 90, 90]
          occme = [29, 90, 59]
          phobme = [59, 69, 80]

          placement = np.arange(3)
          width = 0.25

          bar(placement, occme, width, color='teal', hold=True, label = "Occupy")
          bar([p+width for p in placement], privme, width, color='grey', hold=True, label = "Privilege"
          bar([p+width*2 for p in placement], phobme, width, color='b', hold=True, label = "Phobic" )
          ylabel("Classifier Percent Accuracy")
          title("Classifier Accuracy with Bag of Words and POS Bag")
          plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
          xticks(placement + width, ('SVM', 'Logistic Regression', "Naive Bayes"))

Out[150]: ([<matplotlib.axis.XTick at 0x20f797b8>,
            <matplotlib.axis.XTick at 0x20f6deb8>,
            <matplotlib.axis.XTick at 0x2148fcf8>],
           <a list of 3 Text xticklabel objects>)
```


Classifier Accuracy with Bag of Words and POS Bag
Classifier Percent Accuracy
90
80
70
60
50
40
30
20
10
0
SVM
Logistic Regression
Naive Bayes
Occupy
Privilege
Phobic

In [ ]: