Prototyping semi-supervised

May 3, 2017

1 Setup

In [7]:
```
import tsvopener
import pandas as pd
import numpy as np
from nltk import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from scipy.sparse import csr_matrix, vstack
from sklearn.semi_supervised import LabelPropagation, LabelSpreading
```

```
regex_categorized = tsvopener.open_tsv("categorized.tsv")
human_categorized = tsvopener.open_tsv("human_categorized.tsv")

# Accuracy Check
#
# match = 0
# no_match = 0
# for key in human_categorized:
#     if human_categorized[key] == regex_categorized[key]:
#         match += 1
#     else:
#         no_match += 1
#
# print("accuracy of regex data in {} human-categorized words".format(
#             len(human_categorized)))
# print(match/(match+no_match))
#
# accuracy of regex data in 350 human-categorized words
# 0.7857142857142857
```

2 Prepare Vectors

In [8]:
```
# set up targets for the human-categorized data
targets = pd.DataFrame.from_dict(human_categorized, 'index')
targets[0] = pd.Categorical(targets[0])
targets['code'] = targets[0].cat.codes
# form: | word (label) | language | code (1-5)

tmp_dict = {}
for key in human_categorized:
    tmp_dict[key] = tsvopener.etymdict[key]
supervised_sents = pd.DataFrame.from_dict(tmp_dict, 'index')

all_sents = pd.DataFrame.from_dict(tsvopener.etymdict, 'index')
vectorizer = CountVectorizer(stop_words='english', max_features=10000)
all_sents.index.get_loc("anyways (adv.)")
```

Out[8]:
```
36478
```

In [9]:
```
# vectorize the unsupervised vectors.

vectors = vectorizer.fit_transform(all_sents.values[:,0])

print(vectors.shape)
# supervised_vectors = vectorizer.fit_transform(supervised_data.values[:,0]
```

```
(45723, 10000)
```

In [10]:
```
# add labels

 # initialize to -1
 all_sents['code'] = -1


 supervised_vectors = csr_matrix((len(human_categorized),
                                  vectors.shape[1]),
                                 dtype=vectors.dtype)

 j = 0
 for key in supervised_sents.index:
     all_sents.loc[key]['code'] = targets.loc[key]['code']
     i = all_sents.index.get_loc(key)
     supervised_vectors[j] = vectors[i]
     j += 1



 # supervised_vectors = csr_matrix((len(human_categorized),
```

```
#                                       unsupervised_vectors.shape[1]),
#                                      dtype=unsupervised_vectors.dtype)

# j = 0
# for key in supervised_data.index:
#     i = unsupervised_data.index.get_loc(key)
#     supervised_vectors[j] = unsupervised_vectors[i]
#     j += 1



all_sents.loc['dicky (n.)']
```

```
/home/trevor/anaconda3/envs/etym/lib/python3.5/site-packages/ipykernel/__main__.py:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/i
/home/trevor/anaconda3/envs/etym/lib/python3.5/site-packages/scipy/sparse/compresse
  SparseEfficiencyWarning)
```

Out[10]:
```
0       "detached shirt front," 1811; "a small bird," ...
code                                                   -1
Name: dicky (n.), dtype: object
```

In [ ]:

3 Use Scikit's semisupervised learning

There are two semisupervised methods that scikit has. Label Propagation and Label Spreading. The difference is in how they regularize.

In [23]:
```
num_points = 1000
num_test = 50

x = vstack([vectors[:num_points], supervised_vectors]).toarray()
t = all_sents['code'][:num_points].append(targets['code'])

x_test = x[-num_test:]
t_test = t[-num_test:]
x = x[:-num_test]
t = t[:-num_test]

label_prop_model = LabelSpreading(kernel='knn')
from time import time

print("fitting model")
timer_start = time()
```

```
        label_prop_model.fit(x, t)
        print("runtime: %0.3fs" % (time()-timer_start))
```

```
fitting model
runtime: 409.998s
```

In [24]:
```
print("done!")

        # unsupervised_data['code'].iloc[:1000]
```

```
done!
```

In [11]:
```
import pickle

        # with open("classifiers/labelspreading_knn_all_but_100.pkl", 'bw') as wri
        #     pickle.dump(label_prop_model, writefile)
```

In [25]:
```
import smtplib

        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login("trevortds3@gmail.com", "Picardy3")

        msg = "Job's done!"
        server.sendmail("trevortds3@gmail.com", "trevortds@gmail.com", msg)
        server.quit()
```

Out[25]:
```
(221, b'2.0.0 closing connection 17sm495479otj.30 - gsmtp')
```

In [15]:
```
targets
```

Out[15]:

	0	code
keg (n.)	Norse	4
Ganymede	Greek	2
raw (adj.)	English	0
handle (n.)	English	0
cardamom (n.)	French	1
bravo	Other	5
wicket (n.)	French	1
girandole (n.)	French	1
deputize (v.)	French	1
Cambodia	Other	5
demeaning (adj.)	English	0
fillet (v.)	French	1
jeunesse doree (n.)	French	1
concurring (adj.)	Latin	3
transaction (n.)	French	1

survival (n.)	French	1
Angevin	French	1
acme (n.)	Greek	2
anamorphic (adj.)	Greek	2
assortment (n.)	French	1
noli me tangere	Latin	3
Lernaean	Latin	3
lark (v.)	English	0
Toussaint (n.)	French	1
marble (v.)	French	1
wooly (adj.)	English	0
serving (n.)	French	1
hereafter (adv.)	English	0
phi	Greek	2
constable (n.)	French	1
...	...	...
metric (adj.)	French	1
clinic (n.)	French	1
bracken (n.)	Norse	4
over-excitement (n.)	French	1
patho-	Greek	2
atrophy (v.)	French	1
gaydar (n.)	English	0
wishbone (n.)	English	0
latter (adv.)	English	0
world war (n.)	English	0
oocyte (n.)	Greek	2
puddinghead (n.)	English	0
naysayer (n.)	French	1
externalize (v.)	French	1
pyrotechnic (adj.)	Greek	2
snowfall (n.)	English	0
setter (n.)	English	0
flageolet (n.)	French	1
piracy (n.)	Latin	3
Sahel	Other	5
dulcimer (n.)	French	1
whoever (pron.)	English	0
geo-	Greek	2
Mata Hari	Other	5
rhotacism (n.)	Latin	3
sparkle (n.)	English	0
imbue (v.)	Latin	3
empathetic (adj.)	Greek	2
thermal (adj.)	French	1
doorway (n.)	English	0

[350 rows x 2 columns]

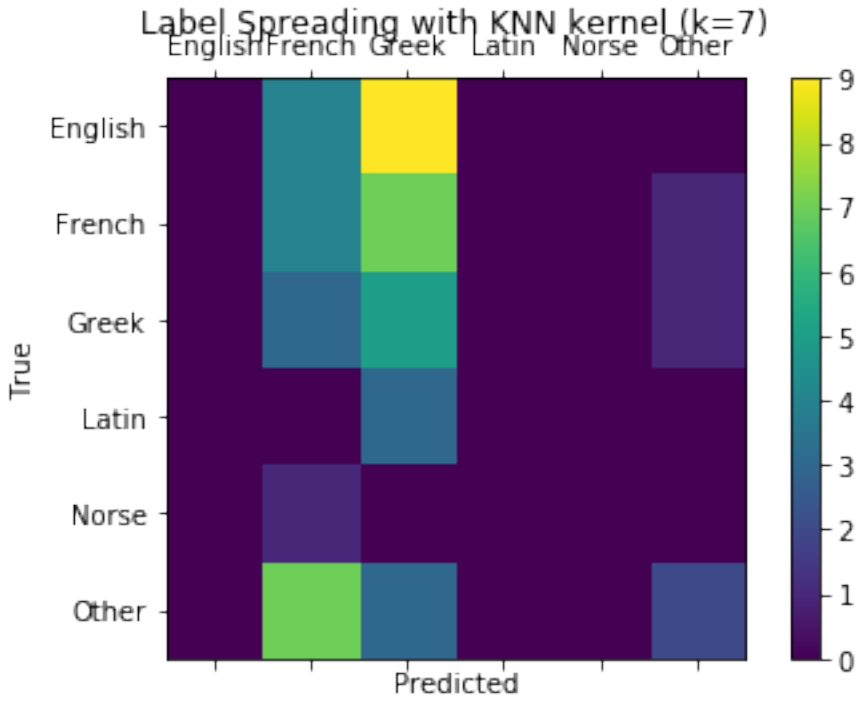
4 Measuring effectiveness.

In [26]:
```
from sklearn.metrics import precision_score, accuracy_score, f1_score, rec
```

```
t_pred = label_prop_model.predict(x_test)

print("Metrics based on 50 hold-out points")

print("Macro")
print("accuracy: %f" % accuracy_score(t_test, t_pred))
print("precision: %f" % precision_score(t_test, t_pred, average='macro'))
print("recall: %f" % recall_score(t_test, t_pred, average='macro'))
print("f1: %f" % f1_score(t_test, t_pred, average='macro'))
print("\n\nMicro")
print("accuracy: %f" % accuracy_score(t_test, t_pred))
print("precision: %f" % precision_score(t_test, t_pred, average='micro'))
print("recall: %f" % recall_score(t_test, t_pred, average='micro'))
print("f1: %f" % f1_score(t_test, t_pred, average='micro'))

from sklearn import metrics
import matplotlib.pyplot as pl

labels = ["English", "French", "Greek", "Latin","Norse", "Other"]
labels_digits = [0, 1, 2, 3, 4, 5]
cm = metrics.confusion_matrix(t_test, t_pred, labels_digits)

fig = pl.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
pl.title("Label Spreading with KNN kernel (k=7)")
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
pl.xlabel('Predicted')
pl.ylabel('True')

pl.show()
```

```
Metrics based on 100 hold-out points
Macro
accuracy: 0.220000
precision: 0.149285
recall: 0.175926
f1: 0.130974
```

```
Micro
accuracy: 0.220000
precision: 0.220000
recall: 0.220000
f1: 0.220000


/home/trevor/anaconda3/envs/etym/lib/python3.5/site-packages/sklearn/metrics/classi
  'precision', 'predicted', average, warn_for)
/home/trevor/anaconda3/envs/etym/lib/python3.5/site-packages/sklearn/metrics/classi
  'precision', 'predicted', average, warn_for)
```



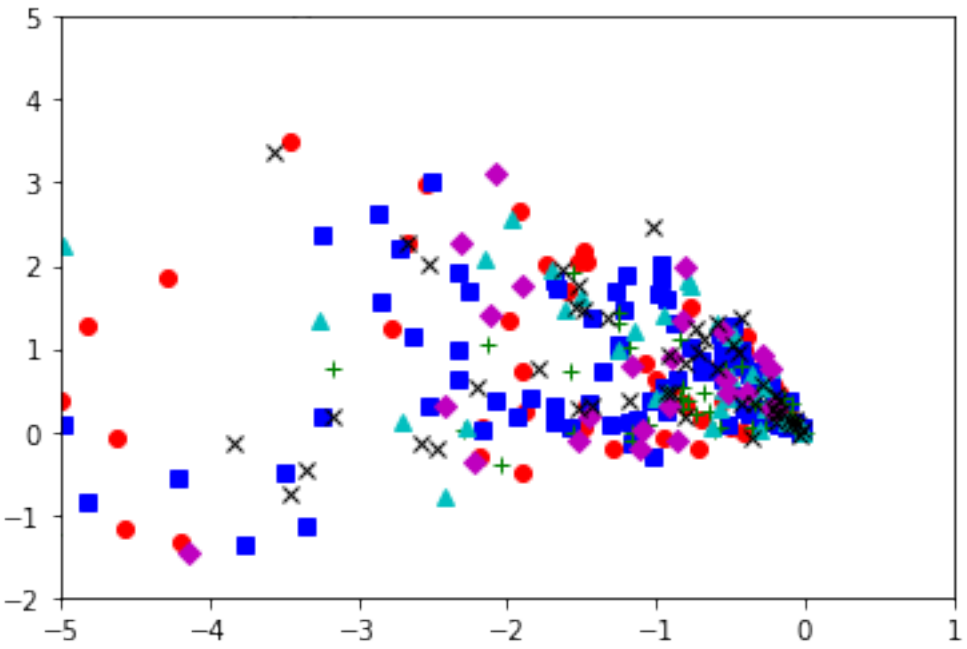
5 PCA: Let's see what it looks like

Performing PCA

```
In [11]: supervised_vectors
```

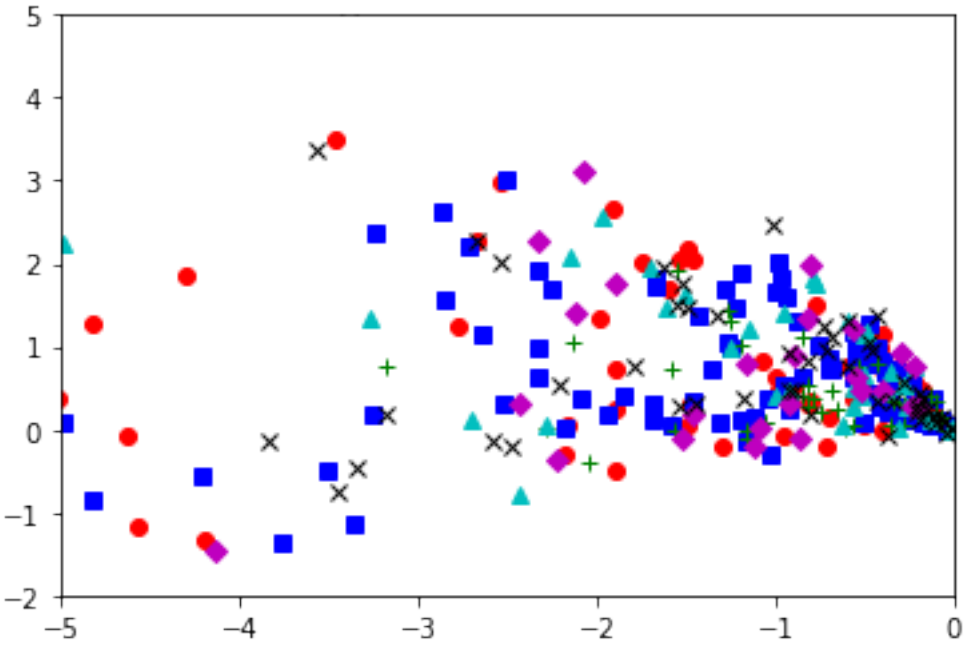
```
Out[11]: <350x10000 sparse matrix of type '<class 'numpy.int64'>'
                with 9602 stored elements in Compressed Sparse Row format>
```

```
In [13]: import matplotlib.pyplot as pl

         u, s, v = np.linalg.svd(supervised_vectors.toarray())
         pca = np.dot(u[:,0:2], np.diag(s[0:2]))
```




```
In [15]: english = np.empty((0,2))
         french = np.empty((0,2))
         greek = np.empty((0,2))
         latin = np.empty((0,2))
         norse = np.empty((0,2))
         other = np.empty((0,2))

         for i in range(pca.shape[0]):
             if targets[0].iloc[i] == "English":
                 english = np.vstack((english, pca[i]))
             elif targets[0].iloc[i] == "French":
                 french = np.vstack((french, pca[i]))
             elif targets[0].iloc[i] == "Greek":
                 greek = np.vstack((greek, pca[i]))
             elif targets[0].iloc[i] == "Latin":
                 latin = np.vstack((latin, pca[i]))
             elif targets[0].iloc[i] == "Norse":
                 norse = np.vstack((norse, pca[i]))
             elif targets[0].iloc[i] == "Other":
                 other = np.vstack((other, pca[i]))

         pl.plot( english[:,0], english[:,1], "ro",
                   french[:,0],  french[:,1], "bs",
                    greek[:,0],   greek[:,1], "g+",
                    latin[:,0],   latin[:,1], "c^",
                    norse[:,0],   norse[:,1], "mD",
```

```
                other[:,0],    other[:,1], "kx")
pl.axis([-5,0,-2, 5])
pl.show()
```




```
In [17]: print (s)
```

```
[  5.91620224e+01   2.71903266e+01   2.25373321e+01   2.18548378e+01
   2.05293711e+01   2.02552813e+01   1.90754512e+01   1.85991696e+01
   1.81873986e+01   1.80134132e+01   1.72471862e+01   1.68548772e+01
   1.65085407e+01   1.62650006e+01   1.57953228e+01   1.54611529e+01
   1.50303486e+01   1.48647240e+01   1.42207480e+01   1.40830743e+01
   1.38100719e+01   1.36810377e+01   1.34711118e+01   1.29794841e+01
   1.29328253e+01   1.26212845e+01   1.25284216e+01   1.22711388e+01
   1.22234562e+01   1.20992262e+01   1.19344953e+01   1.18256517e+01
   1.16621712e+01   1.15169454e+01   1.11559128e+01   1.11172564e+01
   1.09965146e+01   1.07834035e+01   1.06354416e+01   1.05598111e+01
   1.05040540e+01   1.03044157e+01   1.00384970e+01   9.95060504e+00
   9.80957830e+00   9.62082994e+00   9.59783806e+00   9.55042494e+00
   9.40766998e+00   9.27471794e+00   9.12442460e+00   9.00725264e+00
   8.94174012e+00   8.89634153e+00   8.86327938e+00   8.73631683e+00
   8.63344898e+00   8.56110426e+00   8.43019754e+00   8.37597463e+00
   8.33290257e+00   8.27182220e+00   8.23673133e+00   8.12892926e+00
   8.05098156e+00   8.01639554e+00   7.89447701e+00   7.85036818e+00
   7.75353363e+00   7.70371203e+00   7.69309697e+00   7.59309821e+00
   7.53870605e+00   7.42894652e+00   7.36593410e+00   7.33917439e+00
   7.30685048e+00   7.23261164e+00   7.18562750e+00   7.16422048e+00
```

7.07553708e+00 7.06647784e+00 7.00851735e+00 6.95249135e+00
6.87706673e+00 6.86320591e+00 6.85258130e+00 6.73528203e+00
6.71341256e+00 6.62620067e+00 6.61189382e+00 6.56551907e+00
6.48725053e+00 6.47353990e+00 6.44774014e+00 6.42189497e+00
6.34945126e+00 6.33635203e+00 6.29718001e+00 6.25692748e+00
6.19701173e+00 6.17973520e+00 6.15567560e+00 6.12907840e+00
6.08100714e+00 6.06000577e+00 6.03287749e+00 6.02933807e+00
5.99912320e+00 5.97827993e+00 5.91807281e+00 5.89696472e+00
5.85928901e+00 5.83465375e+00 5.81825110e+00 5.76785173e+00
5.69884611e+00 5.68441884e+00 5.67886008e+00 5.66503104e+00
5.64118308e+00 5.62685802e+00 5.62328862e+00 5.54333149e+00
5.51022672e+00 5.50212372e+00 5.45635845e+00 5.42840128e+00
5.41055251e+00 5.39485463e+00 5.37178423e+00 5.34700924e+00
5.33119914e+00 5.30342689e+00 5.28174842e+00 5.26575457e+00
5.23834984e+00 5.20694135e+00 5.19020006e+00 5.16171577e+00
5.15307331e+00 5.11366306e+00 5.08951824e+00 5.08043932e+00
5.05632805e+00 5.03951880e+00 5.02170609e+00 4.99330871e+00
4.97455397e+00 4.96131269e+00 4.90657807e+00 4.90421133e+00
4.88360745e+00 4.87805323e+00 4.84127903e+00 4.83383211e+00
4.79793617e+00 4.79084463e+00 4.74435725e+00 4.72977315e+00
4.72633702e+00 4.69921667e+00 4.67594537e+00 4.64762834e+00
4.62283660e+00 4.61123143e+00 4.59333483e+00 4.55667800e+00
4.55487119e+00 4.54252515e+00 4.51858068e+00 4.47135159e+00
4.46751138e+00 4.44309841e+00 4.42274337e+00 4.41170017e+00
4.38343403e+00 4.36344550e+00 4.34710623e+00 4.31680208e+00
4.30357336e+00 4.29415007e+00 4.27494983e+00 4.25288321e+00
4.24258225e+00 4.21292455e+00 4.18465889e+00 4.17666500e+00
4.16848831e+00 4.13179108e+00 4.10662428e+00 4.09367116e+00
4.08276950e+00 4.06962523e+00 4.05648628e+00 4.02270282e+00
4.00157776e+00 3.98486000e+00 3.97521433e+00 3.95748808e+00
3.94071036e+00 3.92832089e+00 3.92400654e+00 3.91596423e+00
3.87691156e+00 3.86760988e+00 3.84023574e+00 3.83699536e+00
3.80326425e+00 3.78249742e+00 3.75428849e+00 3.73462003e+00
3.70710567e+00 3.69914220e+00 3.68744123e+00 3.66623141e+00
3.65410376e+00 3.62376951e+00 3.61985574e+00 3.59945490e+00
3.56853079e+00 3.54631093e+00 3.54045391e+00 3.52498211e+00
3.49025551e+00 3.47486359e+00 3.46983201e+00 3.45639446e+00
3.43247778e+00 3.42303538e+00 3.41622757e+00 3.38370225e+00
3.36899284e+00 3.35735153e+00 3.33617643e+00 3.31651429e+00
3.30893375e+00 3.28222948e+00 3.26545621e+00 3.25579743e+00
3.23714510e+00 3.21889205e+00 3.19853411e+00 3.19054881e+00
3.17404164e+00 3.16397576e+00 3.14724600e+00 3.12090826e+00
3.09682950e+00 3.09109158e+00 3.08198848e+00 3.07408811e+00
3.05489219e+00 3.00045712e+00 2.98562464e+00 2.97663690e+00
2.96080758e+00 2.94171719e+00 2.92512140e+00 2.91339012e+00
2.90456883e+00 2.89698888e+00 2.88588297e+00 2.87435774e+00
2.84774494e+00 2.83427309e+00 2.82253109e+00 2.81218591e+00
2.79407108e+00 2.78091722e+00 2.77164313e+00 2.75786737e+00

```
2.73232181e+00   2.72205267e+00   2.68973589e+00   2.67816897e+00
2.66768897e+00   2.65612647e+00   2.63527825e+00   2.62218115e+00
2.60899033e+00   2.60179147e+00   2.58519333e+00   2.57398729e+00
2.55319231e+00   2.54707318e+00   2.51525775e+00   2.50928206e+00
2.49798107e+00   2.47752596e+00   2.46082405e+00   2.44789996e+00
2.43781208e+00   2.42337148e+00   2.37877014e+00   2.36397765e+00
2.34423063e+00   2.32521781e+00   2.30443806e+00   2.28767667e+00
2.25974349e+00   2.25225851e+00   2.23069208e+00   2.20879433e+00
2.19818227e+00   2.15452837e+00   2.13003021e+00   2.10114354e+00
2.08408874e+00   2.06706333e+00   2.03305891e+00   2.00453592e+00
1.99422035e+00   1.98176476e+00   1.96403012e+00   1.95435470e+00
1.89485616e+00   1.85497403e+00   1.80593168e+00   1.75025015e+00
1.72640568e+00   1.71779958e+00   1.71208774e+00   1.69306229e+00
1.67024606e+00   1.65664206e+00   1.61255136e+00   1.59659905e+00
1.48568398e+00   1.43589975e+00   1.36775590e+00   1.33071770e+00
1.30151167e+00   1.28251284e+00   1.24081899e+00   1.22199771e+00
1.20023909e+00   1.18018785e+00   1.15772257e+00   1.11618851e+00
1.09146571e+00   1.06876030e+00   1.01863914e+00   9.89895736e-01
6.77936796e-01   5.63668345e-01   2.62730405e-14   4.40975938e-15
1.55993787e-15   1.48858648e-15]
```