Homework 4

February 25, 2019

0.1 Worksheet 8 - Generative models 3

Problem 3:

Part a)

Unpacked and sorted through the directories, have 20 classifications of news types, which are informed by the directory hierarchy

Part b)

Leverage data-set and hierarchy on scikit-learn. Links to the same directory as specified in the homework. A trainling label will link fo the lableed value according to the directory that it lies in, i.e.: alt.atheism 1, etc.

```
In [11]: from sklearn.datasets import fetch_20newsgroups
         from sklearn.feature_extraction.text import TfidfVectorizer
         # Remove strong identifiers of article category
         newsgroups_train = fetch_20newsgroups(subset='train',remove=('headers', 'footers', 'quo
         # Remove strong identifiers of article category
         newsgroups_test = fetch_20newsgroups(subset='test',remove=('headers', 'footers', 'quote
```

```
In [2]: print(newsgroups_train.filenames.shape)
        print(newsgroups_test.filenames.shape)
```

```
(11314,)
(7532,)
```

We have 11,314 documents of training data and 7,532 documents of test data.

Part c)

```
In [3]: length = newsgroups_train.filenames.shape[0]
```

```
In [4]: import numpy as np
        unique, counts = np.unique(newsgroups_train.target, return_counts=True)
```

```
In [5]: import pandas as pd
        prior_prob = pd.DataFrame({'class':unique, 'prior_prob':counts/length})
```

```
In [6]: # Fraction of total documents that belong to each class. Appear to be less on the last c
        # Appears that the class is transformed from the range 0-19 as opposed to 1-20.
        prior_prob
```

```
Out[6]:     class  prior_prob
        0       0    0.042425
        1       1    0.051617
        2       2    0.052236
        3       3    0.052148
        4       4    0.051087
        5       5    0.052413
        6       6    0.051706
        7       7    0.052501
        8       8    0.052855
        9       9    0.052766
        10     10    0.053032
        11     11    0.052590
        12     12    0.052236
        13     13    0.052501
        14     14    0.052413
        15     15    0.052943
        16     16    0.048259
        17     17    0.049850
        18     18    0.041100
        19     19    0.033322
```

```
In [8]: vocab = {}
        reverse_vocab = {}
        count = 0
        a = open('./vocabulary.txt', 'r')
        for v in a:
            val = v.strip()
            vocab[val] = count
            reverse_vocab[count] = val
            count += 1
```

```
In [252]: vocab['baseball']
```

```
Out[252]: 1816
```

```
In [215]: # Vectorize each training document, using the vocabulary document
          vectorizer = TfidfVectorizer(strip_accents='unicode', decode_error = 'ignore', stop_wo
          vectors = vectorizer.fit_transform(newsgroups_train.data)
          vectors.shape
```

```
Out[215]: (11314, 61188)
```

```
In [216]: # Vectorize the test document, using the vocabulary document
          vectors_test = vectorizer.fit_transform(newsgroups_test.data)
          vectors_test.shape
```

```
Out[216]: (7532, 61188)
```

Part d) Used a different smoothing constant than 1, 1 did not perform as well. MultinomialNB uses logs inherently.

```
In [223]: from sklearn.naive_bayes import MultinomialNB
          from sklearn import metrics
          clf = MultinomialNB(alpha=0.046)
          clf.fit(vectors, newsgroups_train.target)
          # Naive bayes uses prior probability distributions from above
```

```
Out[223]: MultinomialNB(alpha=0.046, class_prior=None, fit_prior=True)
```

Part e)

```
In [224]: # Predictions:
          pred = clf.predict(vectors_test)
```

```
In [225]: print('The model is', round(metrics.accuracy_score(newsgroups_test.target, pred)*100),
          print('The model has an error rate of', round((1- metrics.accuracy_score(newsgroups_te
```

```
The model is 70.0 % accurate
The model has an error rate of 30.0 %
```

0.2 Worksheet 9 - Clustering

Problem 1:

```
In [6]: f = open('./Animals_with_Attributes/Features/README-features.txt', 'r')
        file_contents = f.read()
        print (file_contents)
        f.close()
```

```
=========================================================================
   Pre-Extracted Image Features for the Animals_with_Attributes Dataset
=========================================================================

Names and IDs of all classes are in the file
- classes.txt

Predicate names andIDs are in the file
- predicates.txt

Training and test classnames for the attribute-based classifier are in
- trainclasses.txt, testclasses.txt

The class<->attribute matrix is given in three formats.
- predicate-matrix-numeric.txt (positive numeric entries, some missing
  entries encoded as -1)
- predicate-matrix-binary.txt (binarized with mean of feature, missing
```

set to 0)

- predicate-matrix.png (PNG image file for visual inspection)

Note that the entries are in the order of the names/predicates files, *not* alphabetically.

There are 6 feature representations:

- cq: (global) color histogram (1x1 + 2x2 + 4x4 spatial pyramid, 128 bins each, each histogram L1-normalized)
- lss[1]: local self similarity (2000 entry codebook, raw bag-of-visual-word counts)
- phog[2]: histogram of oriented gradients (1x1 + 2x2 + 4x4 spatial pyramid, 12 bins each, each histogram L1-normalized or all zero)
- rgsift[3]: rgSIFT descriptors (2000 entry codebook, bag-of-visual-word counts, L1-normalized)
- sift[4]: SIFT descriptors (2000 entry codebook, raw bag-of-visual-word counts)
- surf[5]: SUFT descriptors (2000 entry codebook, raw bag-of-visual-word counts)

Each file consists of one sample per line in ASCII format.
All representations have non-negative entries.

Instructions for fixed-split Attribute-Based Classification

- train using all examples of all 40 classes from "trainclasses.txt"
- test on all examples of 10 classes specified in the "testclasses.txt"

Instructions for CV-like Multi-Class Classification

Follow a protocol similar to Caltech256:

- choose N_train \in {5,10,15,20,25,30,40,50} and set N_test=25
- train on N_train random training examples from each class
- test on N_test random examples out of the remaining images from each class
- calculate the mean of the confusion matrix (class averaged accuracy)
- report averaged results for 10-times this procedure

References

[1] E. Shechtman, and M. Irani: "Matching Local Self-Similarities across Images and Videos", CVPR 2007.

[2] A. Bosch, A. Zisserman, and X. Munoz: "Representing shape with
a spatial pyramid kernel", CIVR 2007.

[3] Koen E. A. van de Sande, Theo Gevers and Cees G. M. Snoek:
"Evaluation of Color Descriptors for Object and Scene
Recognition", CVPR 2008.

[4] D. G. Lowe, "Distinctive Image Features from Scale-Invariant
Keypoints", IJCV 2004.

[5] H. Bay, T. Tuytelaars, and L. Van Gool: "SURF: Speeded Up
Robust Features", ECCV 2006.

Problem 2:

```
In [7]: # Different animal classes
        f = open('./Animals_with_Attributes/classes.txt', 'r')
        classes_str = f.read()
        print (classes_str)
        f.close()
```

```
 1        antelope
 2        grizzly+bear
 3        killer+whale
 4        beaver
 5        dalmatian
 6        persian+cat
 7        horse
 8        german+shepherd
 9        blue+whale
10        siamese+cat
11        skunk
12        mole
13        tiger
14        hippopotamus
15        leopard
16        moose
17        spider+monkey
18        humpback+whale
19        elephant
20        gorilla
21        ox
22        fox
23        sheep
24        seal
25        chimpanzee
```

```
   26      hamster
   27      squirrel
   28      rhinoceros
   29      rabbit
   30      bat
   31      giraffe
   32      wolf
   33      chihuahua
   34      rat
   35      weasel
   36      otter
   37      buffalo
   38      zebra
   39      giant+panda
   40      deer
   41      bobcat
   42      pig
   43      lion
   44      mouse
   45      polar+bear
   46      collie
   47      walrus
   48      raccoon
   49      cow
   50      dolphin
```

```
In [8]: # Different available features
        f = open('./Animals_with_Attributes/predicates.txt', 'r')
        features_str = f.read()
        print (features_str)
        f.close()
```

```
    1      black
    2      white
    3      blue
    4      brown
    5      gray
    6      orange
    7      red
    8      yellow
    9      patches
   10      spots
   11      stripes
   12      furry
   13      hairless
   14      toughskin
```

15	big
16	small
17	bulbous
18	lean
19	flippers
20	hands
21	hooves
22	pads
23	paws
24	longleg
25	longneck
26	tail
27	chewteeth
28	meatteeth
29	buckteeth
30	strainteeth
31	horns
32	claws
33	tusks
34	smelly
35	flys
36	hops
37	swims
38	tunnels
39	walks
40	fast
41	slow
42	strong
43	weak
44	muscle
45	bipedal
46	quadrapedal
47	active
48	inactive
49	nocturnal
50	hibernate
51	agility
52	fish
53	meat
54	plankton
55	vegetation
56	insects
57	forager
58	grazer
59	hunter
60	scavenger
61	skimmer
62	stalker

```
63        newworld
64        oldworld
65        arctic
66        coastal
67        desert
68        bush
69        plains
70        forest
71        fields
72        jungle
73        mountains
74        ocean
75        ground
76        water
77        tree
78        cave
79        fierce
80        timid
81        smart
82        group
83        solitary
84        nestspot
85        domestic
```

Problem 3:

```
In [9]: classes = ''.join([i for i in classes_str if not i.isdigit()]).split()

In [10]: classes[:10]

Out[10]: ['antelope',
          'grizzly+bear',
          'killer+whale',
          'beaver',
          'dalmatian',
          'persian+cat',
          'horse',
          'german+shepherd',
          'blue+whale',
          'siamese+cat']

In [11]: features = ''.join([i for i in features_str if not i.isdigit()]).split()

In [12]: features[:10]

Out[12]: ['black',
          'white',
          'blue',
          'brown',
          'gray',
          'orange',
          'red',
          'yellow',
          'patches',
          'spots']
```

```
In [13]: animals_data = pd.read_fwf("./Animals_with_Attributes/predicate-matrix-continuous.txt",
         print('The shape of the data is', animals_data.shape)
```

```
The shape of the data is (50, 85)
```

```
In [14]: animal_df = pd.DataFrame(data = animals_data, columns = features)
         animal_df.index = classes
         animal_df.head()
```

```
Out[14]:               black  white  blue  brown   gray  orange  red  yellow  patches  \
         antelope      -1.00  -1.00  -1.0  -1.00  12.34     0.0  0.0     0.0    16.11
         grizzly+bear  39.25   1.39   0.0  74.14   3.75     0.0  0.0     0.0     1.25
         killer+whale  83.40  64.79   0.0   0.00   1.25     0.0  0.0     0.0    68.49
         beaver        19.38   0.00   0.0  87.81   7.50     0.0  0.0     0.0     0.00
         dalmatian     69.58  73.33   0.0   6.39   0.00     0.0  0.0     0.0    37.08

                        spots   ...    water  tree   cave  fierce  timid  smart  \
         antelope        9.19   ...     0.00  0.00   1.23   10.49  39.24  17.57
         grizzly+bear    0.00   ...     7.64  9.79  53.14   61.80  12.50  24.00
         killer+whale   32.69   ...    79.49  0.00   0.00   38.27   9.77  52.03
         beaver          7.50   ...    65.62  0.00   0.00    3.75  31.88  41.88
         dalmatian     100.00   ...     1.25  6.25   0.00    9.38  31.67  53.26

                       group  solitary  nestspot  domestic
         antelope      50.59      2.35      9.70      8.38
         grizzly+bear   3.12     58.64     20.14     11.39
         killer+whale  24.94     15.77     13.41     15.42
         beaver        23.44     31.88     33.44     13.12
         dalmatian     24.44     29.38     11.25     72.71

         [5 rows x 85 columns]
```

```
In [15]: # Import K Means Package
         from sklearn.cluster import KMeans

         # Set k = 10
         km10 = KMeans(n_clusters=10)
         km10.fit(animals_data)
         # Get cluster assignment labels
         labels = km10.labels_
         # Format results as a DataFrame
         results = pd.DataFrame([animal_df.index,labels]).T
         results.columns = ['class', 'cluster']
```

```
In [16]: results.groupby('cluster')['class'].apply(list)
```

```
Out[16]: cluster
         0                 [spider+monkey, gorilla, chimpanzee]
         1                 [hippopotamus, elephant, rhinoceros]
         2                           [grizzly+bear, polar+bear]
         3              [antelope, horse, giraffe, zebra, deer]
         4    [beaver, skunk, mole, hamster, squirrel, rabbi...
         5    [killer+whale, blue+whale, humpback+whale, sea...
         6                                                [bat]
         7            [tiger, leopard, fox, wolf, bobcat, lion]
         8    [moose, ox, sheep, buffalo, giant+panda, pig, ...
         9    [dalmatian, persian+cat, german+shepherd, siam...
         Name: class, dtype: object
```

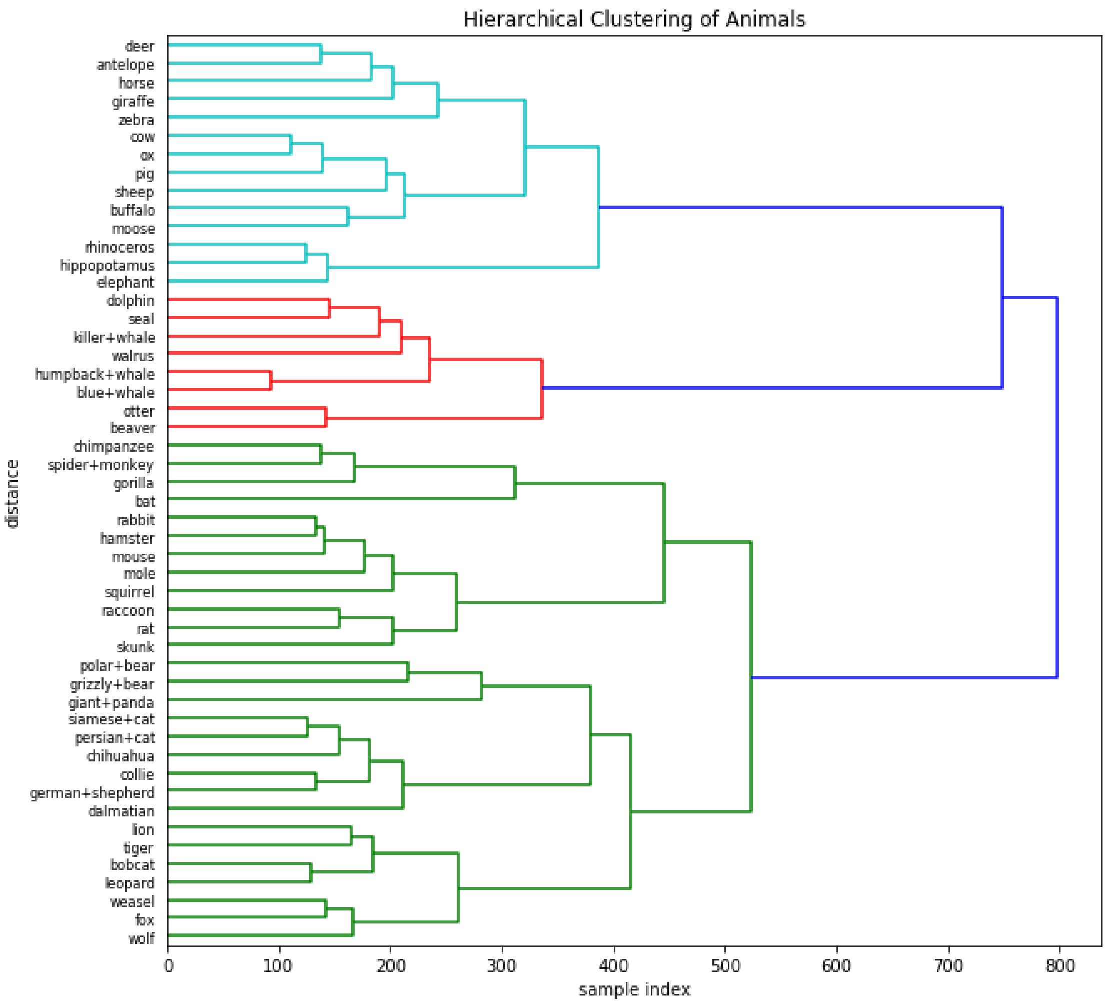
To me, it looks like the clusters make pretty good sense. The large aquatic/land animals are grouped together, the flying animal is alone, the bears are together, and the household pets are grouped together.

Problem 4:

```
In [17]: from scipy.cluster.hierarchy import dendrogram, linkage
         import matplotlib.pyplot as plt
         %matplotlib inline
         from pylab import rcParams
         rcParams['figure.figsize'] = 10,10
```

```
In [18]: HC = linkage(animals_data, 'ward')
```

```
In [20]: plt.title('Hierarchical Clustering of Animals')
         plt.xlabel('sample index')
         plt.ylabel('distance')
         dendrogram(HC, labels= classes, orientation='right')
         plt.show()
```



The hierarchial clusters make sense to me as the larger land animals are grouped together, the smaller land animals are grouped together, and the aquatic animals are grouped together. In these examples, however, I believe the K-means was very comparable, especially when I see the Bat and Monkey family so similar.

0.3 Worksheet 10 - PCA and SVD

Problem 4:

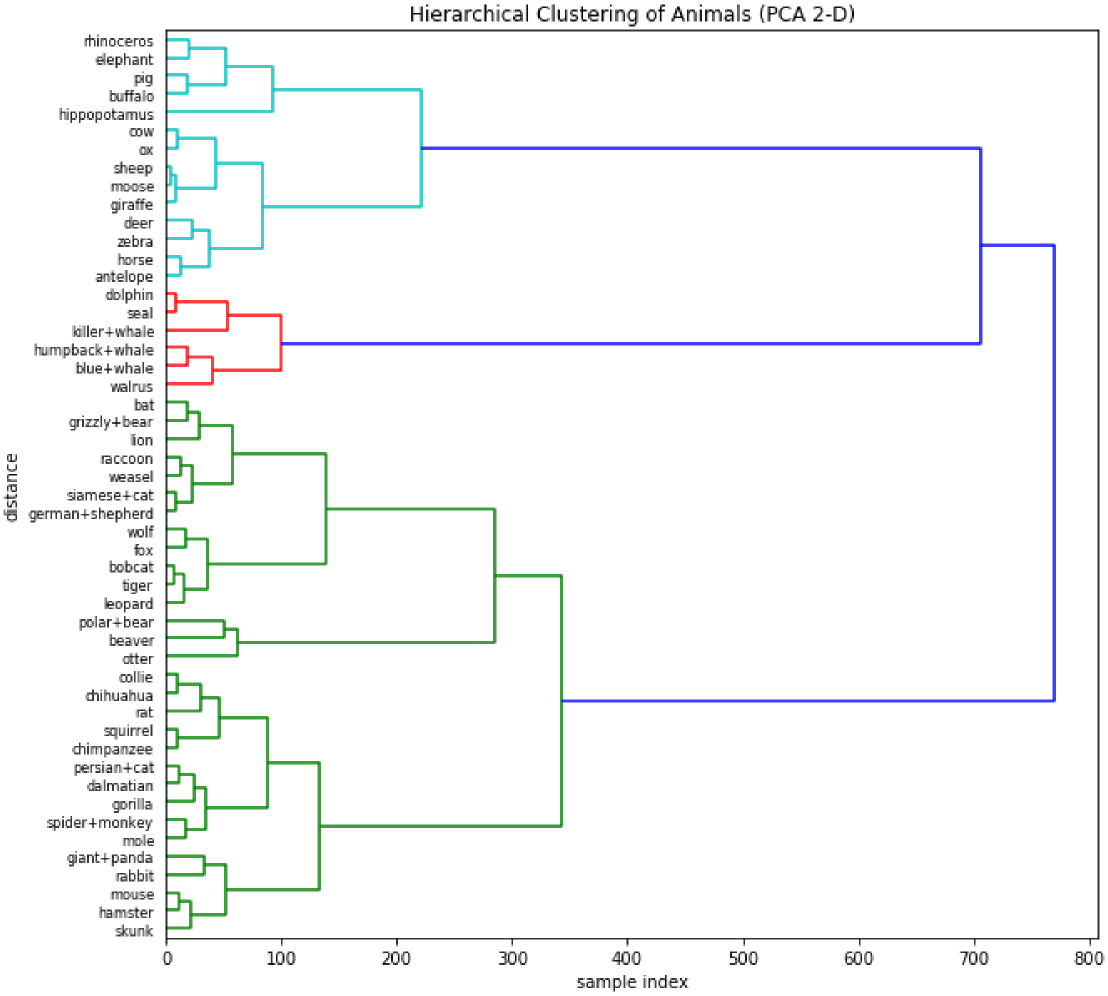
```
In [21]: from sklearn.decomposition import PCA
         pc2 = PCA(2)
         animals_data2d = pc2.fit_transform(animals_data)
         print(animals_data2d.shape)
```

```
(50, 2)
```

```
In [25]: print('Reduced to 2-D dimensionality retained',sum(pc2.explained_variance_ratio_),'of t
Reduced to 2-D dimensionality retained 0.389615467763 of the original datas variance
```

In [26]:

```
# Let us plot the hierarchial clusters again:
HC2 = linkage(animals_data2d, 'ward')
plt.title('Hierarchical Clustering of Animals (PCA 2-D)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(HC2, labels= classes, orientation='right')
plt.show()
```

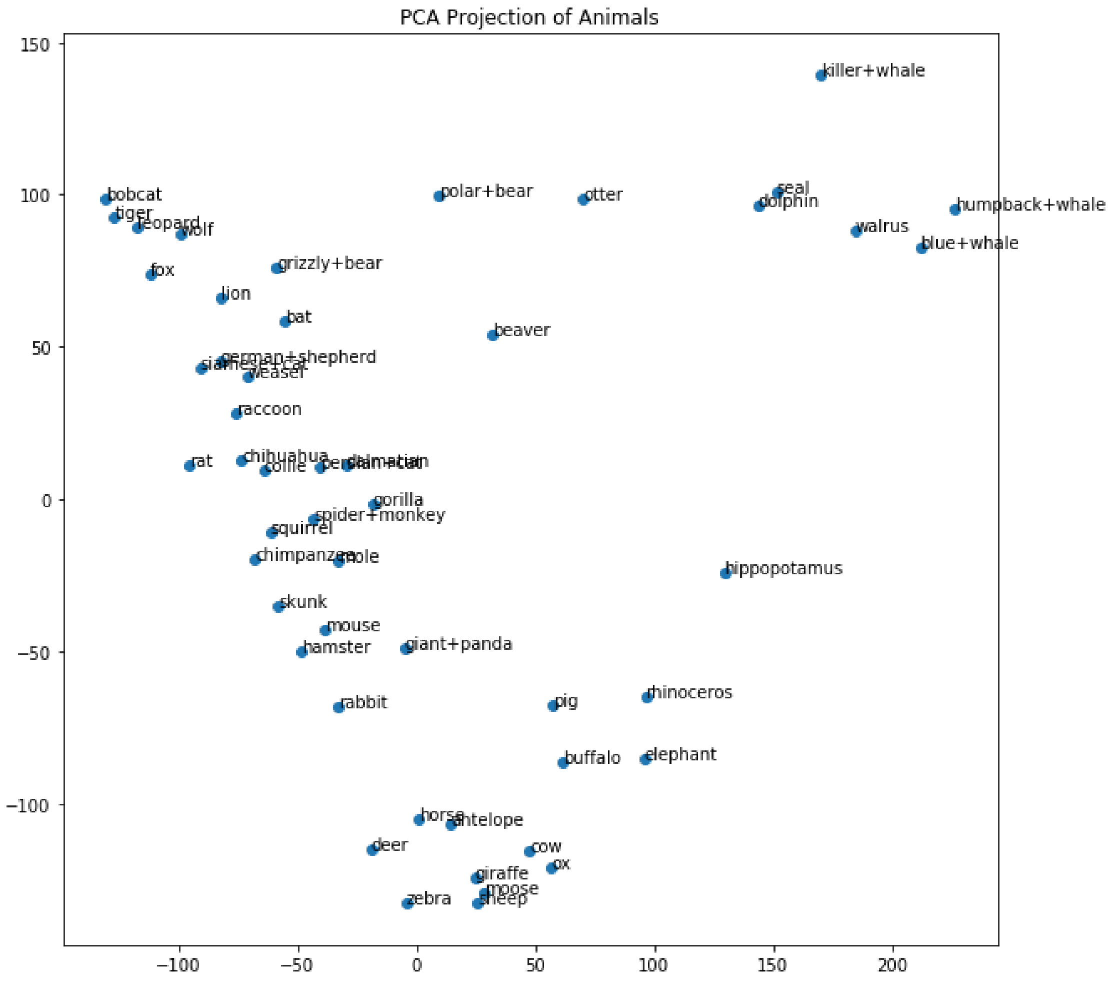


In [27]:

```
# Plot values in 2-D
fig = plt.figure(1, figsize=(10, 10))
ax = fig.add_subplot(111)
for i,point in enumerate(animals_data2d):
    ax.annotate(classes[i], xy=point, xytext=point)
```

```
plt.scatter(animals_data2d[:,0], animals_data2d[:,1])
plt.title('PCA Projection of Animals')
```

```
Out[27]: <matplotlib.text.Text at 0x119db05c0>
```



While it does seem sensible, I think a few higher dimensions may yield more accurate results. This just seems to separate aquatic from non-aquatic animals