image_challenge

August 16, 2020

1 Novoic ML challenge – image data

1.1 Introduction

Welcome to the Novoic ML challenge!

This is an open-ended ML challenge to help us identify exceptional researchers and engineers. The guidance below describes an open-source dataset that you can use to demonstrate your research skills, creativity, coding ability, scientific communication or anything else you think is important to the role.

Before starting the challenge, go ahead and read our CEO's Medium post on what we're looking for in our Research Scientists, Research Engineers and ML Interns. We recommend you spend around three hours on this (more or less if you wish), which you do not have to do in one go. Please make use of any resources you like.

This is the image version of the challenge. Also available are text and audio versions. You can access all three from this GitHub repo.

Best of luck – we're looking forward to seeing what you can do!

1.2 Prepare the data

Copy the dataset to a local directory – this should take around 10 minutes.

[1]:
```
# !mkdir -p data
# !~/utils/gsutil/gsutil -m cp -r gs://novoic-ml-challenge-image-data/* ./data
```

1.3 Data description

The data comprises 17,125 images in jpg format. Each image is of a realistic scene typically containing a number of objects.

There are 20 object classes of interest: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV monitor.

Each image is labelled with one of three numbers for each object class: - -1 (no objects of this class feature in the image) - 1 (at least one object of this class features in the image) - 0 (at least one object of this class features in the image but they are all difficult to recognise)

[2]:
```
import IPython
IPython.display.Image(filename='data/images/2012_004258.jpg')
```

[2]:

[3]:
```
IPython.display.Image(filename='data/images/2008_007739.jpg')
```

[3]:

Each object class file (e.g. aeroplane.txt) contains the name of the image without the extension (e.g. 2008_007739) followed by a space and then the class label (e.g. -1).

For more information about the dataset, see its README.md.

Directory structure:

```
data/
   images/          # dir for jpg files
   aeroplane.txt    # aeroplane object class labels
   bicycle.txt      # bicycle object class labels
   bird.txt         # bird object class labels
   boat.txt         # boat object class labels
   bottle.txt       # bottle object class labels
   bus.txt          # bus object class labels
   car.txt          # car object class labels
   cat.txt          # cat object class labels
   chair.txt        # chair object class labels
   cow.txt          # cow object class labels
   diningtable.txt # dining table object class labels
   dog.txt          # dog object class labels
   horse.txt        # horse object class labels
```

```
motorbike.txt   # motorbike object class labels
person.txt      # person object class labels
pottedplant.txt # potted plant object class labels
sheep.txt       # sheep object class labels
sofa.txt        # sofa object class labels
train.txt       # train object class labels
tvmonitor.txt   # TV monitor object class labels
LICENSE
README.md
```

1.4 The challenge

This is an open-ended challenge and we want to witness your creativity. Some obvious suggestions: - Data exploration/visualization - Binary/multiclass classification - Anomaly detection - Unsupervised clustering - Model explainability

You're welcome to explore one or more of these topics, or do something entirely different.

Create, iterate on, and validate your work in this notebook, using any packages of your choosing.

You can access a GPU via Runtime -> Change runtime type in the toolbar.

1.5 Submission instructions

Once you're done, send this .ipynb notebook (or a link to it hosted on Google Drive/GitHub with appropriate permissions) to talent@novoic.com, ensuring that outputs from cells (text, plots etc) are preserved.

If you haven't applied already, make sure you submit an application first through our job board.

1.6 Your submission

The below sets up TensorFlow as an example but feel free to use any framework you like.

[4]:
```
# The default TensorFlow version on Colab is 1.x. Uncomment the below to use␣
↪TensorFlow 2.x instead.
# %tensorflow_version 2.x
```

[5]:
```
# tried to be creative here, mostly data exploration

# some of the inferences
# labels have a lot of FALSE NEGATIVES
# How -->
```

```
#

# extract anamolies, as dataset has a lot of false negatives, we tried to find␣
 ↪anamolies/outliers/interesting images
# where multiple labels were present
```

[6]:
```
# import tensorflow as tf
# tf.__version__
```

2 Summary

2.1 Why the Image Challenge

I took up the image challenge because of something new in the dataset (that I had not come across before), this was the '0' coded labels. The '0' code signifies that an object is present but it is not (or not easily) visible

2.2 What I did

Explored the dataset, tried to understand it, followed by some clustering to find interesting images!

2.3 Key takeaways

1. There are too many false negatives in the dataset
2. Found some unusual / wierd / interesting images in the dataset
3. Extracted some patterns on the presence of objects in images

[7]:
```
import pathlib
```

2.3.1 I started off my search on the labels coded '0' i.e. the objects that are in the image but not visible

Separating out the images for the '0' coded labels (occluded labels)

```
for file in *.txt; do
 cat ${file##*/} | grep " 0" | awk '{print $1;}' >> "${file%.*}_missing.txt"
done
```

The above script is just creating separate files for the '0' coded labels If it doesn't work for you, it should be simple to understand and modify (or you could script it in python) Personally, I find simple file manipulations easier in shell

[8]:
```
# get the files
missing_files = []
label_files = []
for path in pathlib.Path("./data").iterdir():
    try:
        if len(str(path).split('_')) == 2:
            missing_files.append(path)
        elif str(path).split('.')[1] == "txt":
            label_files.append(path)
    except:
        continue
```

Now let's create a hashmap from an image to its labels

[9]:
```
# similarly for the occluded labels

image_to_labels = {}
occluded_label_to_images = {}

for file in label_files:

    label = str(file).split('/')[1].split('.')[0]
    for i in open(file):
        image, code = i.split()
        if int(code) == 1:
            if image in image_to_labels:
                image_to_labels[image].append(label)
            else:
                image_to_labels[image] = [label]
        if int(code) == 0:

            if label in occluded_label_to_images:
                occluded_label_to_images[label].append(image)
            else:
                occluded_label_to_images[label] = [image]

# extract all labels

all_labels = set()
for image in image_to_labels:
    image_labels = image_to_labels[image]
    all_labels.update(image_labels)
```

a small experiment to understand correlation between two labels being present in an image

Can this give some insight into the 0 coded labels?

[10]:
```
# lets pick a single pair of labels first --> (Bus, Car)
# lets find the images that contain buses and cars
# ...       the images that contain cars but no buses
# ...       the images that contains buses but no cars

both = []
car_no_bus = []
bus_no_car = []

for image, labels in image_to_labels.items():
    if "bus" in labels and "car" in labels:
        both.append(image)
    if "bus" in labels and not "car" in labels:
        bus_no_car.append(image)
    if not "bus" in labels and "car" in labels:
        car_no_bus.append(image)
```

applying bayes theorem

[11]:
```
p_bus_given_car = len(both)/(len(car_no_bus)+len(both))
```

[12]:
```
p_bus_given_car
```

[12]: 0.12144702842377261

[13]:
```
p_car_given_bus = len(both)/(len(bus_no_car)+len(both))
```

[14]:
```
p_car_given_bus
```

[14]: 0.334916864608076

Result: If an image contains a bus, there is a relatively high chance (0.33) of a car being present too
this makes intuitive sense (as they are both seen on the road)

Could this help us find weird / interesting / outlier images?
Images that contain objects which are not naturally seen together *(for eg: an image that has both a bicycle and an aeroplane)*

2.3.2 Let's find out!

We need to compute the bayes conditional probabilities for all label pairs!

[15]:
```
label_pairs = {}

# could have just used combinations(all_labels, 2)
for label1 in all_labels:
    for label2 in all_labels:
```

```
        if label1 >= label2:
            continue

        label_pairs[(label1, label2)] = [[],[],[]]
        values = label_pairs[(label1, label2)]

        for image, labels in image_to_labels.items():

            if label1 in labels and label2 in labels:
                values[0].append(image)

            elif label1 in labels and not label2 in labels:
                values[1].append(image)

            elif not label1 in labels and label2 in labels:
                values[2].append(image)
```

[16]:
```
# helper functions

def get_images_with_label_pair(l1, l2):
    return label_pairs.get((l1,l2),
           label_pairs.get((l2,l1),
           [None, None, None]))[0]

from IPython.display import Image, display

def display_image(image):
    return Image(filename='./data/images/'+image+'.jpg')

def display_images_with_label_pair(l1, l2, count=2):

    # take 2
    return display(*tuple(map(lambda x: display_image(x),␣
↪get_images_with_label_pair(l1,l2)[0:count])))
```

[17]:
```
label_pairs_probs = {}

# compute P(a|b), P(b|a)

for k, v in label_pairs.items():
        label_pairs_probs[k] = (len(v[0])/(len(v[2])+len(v[0])), len(v[0])/
↪(len(v[1]) + len(v[0])))
```

Let's check out some of the label probabilities

[18]:
```
list(label_pairs_probs.items())[0:10]
```

[18]:
```
[(('bicycle', 'bird'), (0.00130718954248366, 0.0018115942028985507)),
 (('bicycle', 'boat'), (0.001968503937007874, 0.0018115942028985507)),
 (('bicycle', 'sheep'), (0.0, 0.0)),
 (('bicycle', 'cat'), (0.002777777777777778, 0.005434782608695652)),
 (('bicycle', 'diningtable'), (0.0055762081784386614, 0.005434782608695652)),
 (('bicycle', 'sofa'), (0.007889546351084813, 0.007246376811594203)),
 (('bicycle', 'bottle'), (0.043909348441926344, 0.05615942028985507)),
 (('bicycle', 'bus'), (0.023752969121140142, 0.018115942028985508)),
 (('bicycle', 'chair'), (0.01519213583556747, 0.030797101449275364)),
 (('bicycle', 'motorbike'), (0.030418250950570342, 0.028985507246376812))]
```

we now have conditional probabilities for each pair of labels
Let's sort them to find the label pairs with the least chances of occuring together

[19]:
```
label_pairs_sorted_on_prob = sorted(label_pairs_probs.items(), key=lambda x:␣
↪x[1][0]*x[1][1], reverse=True)
```

now lets look at some of the labels that rarely occur together

[20]:
```
# potential odd images
oddity_sorted_pairs = list(filter(lambda x: x[1][0] != 0.0␣
↪,label_pairs_sorted_on_prob))[::-1]
```

So these are the label pairs with the lowest possibilities of being together

[21]:
```
oddity_sorted_pairs[0:10]
```

[21]:
```
[(('aeroplane', 'chair'), (0.0008936550491510277, 0.0014925373134328358)),
 (('bus', 'dog'), (0.0007776049766718507, 0.002375296912114014)),
 (('aeroplane', 'bird'), (0.00130718954248366, 0.0014925373134328358)),
 (('bird', 'tvmonitor'), (0.0017391304347826088, 0.00130718954248366)),
 (('bicycle', 'bird'), (0.00130718954248366, 0.0018115942028985507)),
 (('bird', 'diningtable'), (0.0018587360594795538, 0.00130718954248366)),
 (('aeroplane', 'pottedplant'),
  (0.0018975332068311196, 0.0014925373134328358)),
 (('aeroplane', 'motorbike'), (0.0019011406844106464, 0.0014925373134328358)),
 (('cat', 'sheep'), (0.003076923076923077, 0.000925925925925926)),
 (('cat', 'cow'), (0.0033003300330033004, 0.000925925925925926))]
```

[22]:
```
# Change list slice value to see as many images as you want

for (l1, l2), _ in oddity_sorted_pairs[0:1:]:
    print(l1,l2)
    display_images_with_label_pair(l1, l2, count=1)
```

```
aeroplane chair
```

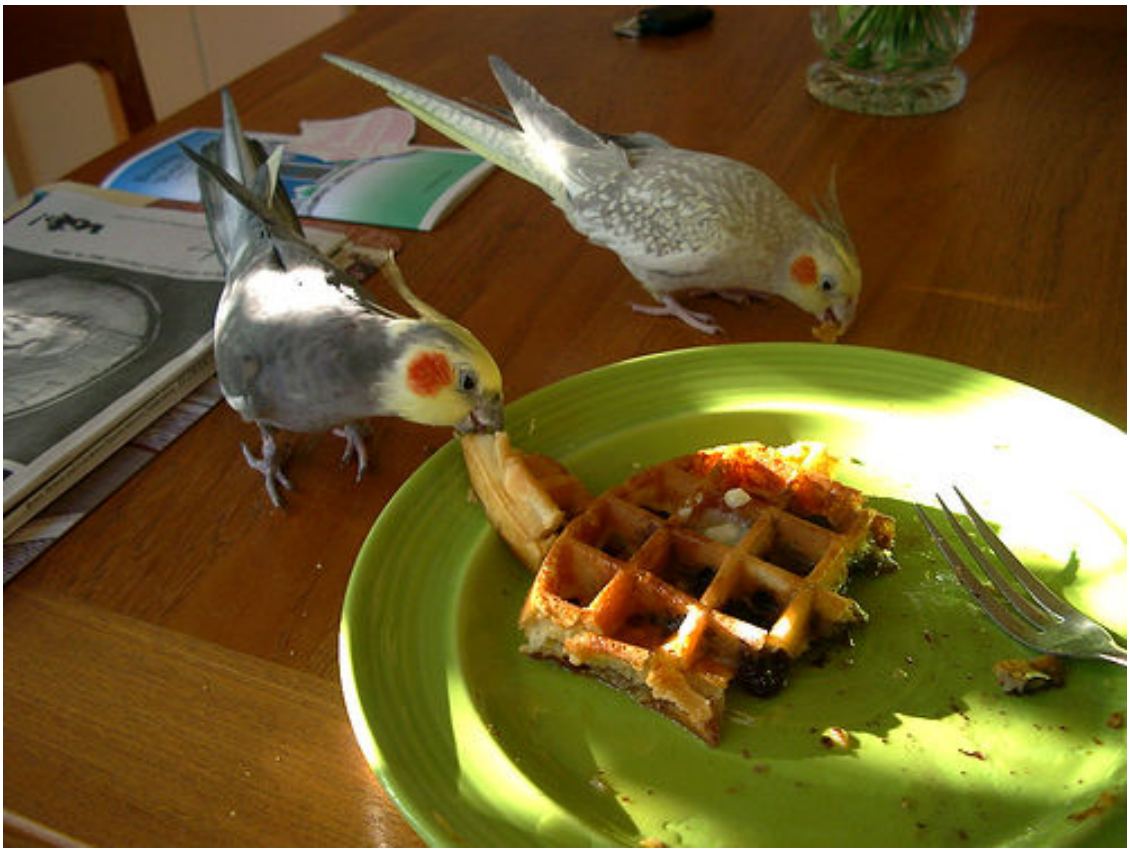
yeah! that's definitely an interesting image
you normally wouldn't see a chair with an aeroplane, and its only in settings like these

[23]:
```
# This is not an interesting image, the dataset doesn't contain enough of these
↪label pairs
display_images_with_label_pair("bus", "dog")
```

[24]:
```
# aeroplanes and birds aren't a safe combination! I can see why they have low␣
↪conditional probabilities
display_images_with_label_pair('aeroplane', 'bird')
```

[25]:
```
# this seems like an anamoly in the labelling because its not a bottle but an␣
↪advertisement of a bottle
# this also makes me want to try the mint chocolate baileys (I love irish␣
↪cream)! xD
display_images_with_label_pair("bottle", "bus")
```

[26]:
```
# this is an image that I wouldn't get to see daily, but its not very␣
↪interesting :(
display_images_with_label_pair('bird', 'diningtable')
```



Okay, so we say a few interesting images, but we also had a few false positives
Can we try to understand the clustering of objects in images? *i.e. groups of objects typically seen together*

Let's find the label pairs with high conditional probabilities

[27]:
```
# the top 5 highly correlated labels
oddity_sorted_pairs[:-6:-1]
```

[27]:
```
[(('chair', 'diningtable'), (0.6263940520446096, 0.30116175156389635)),
 (('bottle', 'diningtable'), (0.2825278810408922, 0.21529745042492918)),
 (('bottle', 'person'), (0.09591387325666748, 0.5552407932011332)),
 (('chair', 'pottedplant'), (0.3301707779886148, 0.1554959785522788)),
 (('chair', 'tvmonitor'), (0.29043478260869565, 0.14924039320822163))]
```

Yeah, I would assume that chairs and dining tables are quite corelated
But, I am interested in anamolies!

Are there images where chairs and dining tables are not present together?

[28]:
```
table_but_no_chairs = label_pairs["chair", "diningtable"][2]
len(table_but_no_chairs)
```

[28]: 201

Wow, 200+ images with tables but no chairs!

Oh wait, the chair might not be visible and so we need to consider the '0' code

[29]:
```
# images containing tables but chairs are indeed not there at all
table_but_surely_no_chairs = set(table_but_no_chairs) -␣
 ↪set(occluded_label_to_images['chair'])
len(table_but_surely_no_chairs)
```

[29]: 153

Wow, 150+ images is also high!
Let's see some images

These are the takeaways:

1. There are images where the chair is visible -- False Negatives (Code 1)
2. There are images where the chair is occluded but definitely there -- False Negatives (Code 0)
3. There are a very few correctly labelled images and they include dining rooms in Japan or restaurants with lounge seating

[30]:
```
# change the slice value to see some images
display(*map(display_image, list(table_but_surely_no_chairs)[1:2]))
```

Let's find the objects that are usually found along with chairs and tables!

[31]:
```
groupings = {}
for image in get_images_with_label_pair("chair", "diningtable"):
    groupings[frozenset(image_to_labels[image])] = groupings.
 ↪get(frozenset(image_to_labels[image]), 0) + 1
for group in sorted(map(lambda x: (x[1],list(x[0])), groupings.items()),␣
 ↪reverse=True)[0:10]:
    print(group[1])
```

```
['chair', 'diningtable']
['person', 'chair', 'diningtable']
['chair', 'pottedplant', 'diningtable']
['bottle', 'chair', 'person', 'diningtable']
['sofa', 'chair', 'diningtable']
['bottle', 'chair', 'diningtable']
['person', 'chair', 'pottedplant', 'diningtable']
['chair', 'cat', 'diningtable']
['sofa', 'chair', 'pottedplant', 'diningtable']
['bottle', 'chair', 'pottedplant', 'diningtable']
```

Yes, the above groups make sense!

we would typically see people, pottedplants, bottles, sofa, and pet cats in a dining room

2.4 Time for some clustering!

2.4.1 We would like to find clusters of labels

This can give insights into common clusters as well as anamolies

[32]:
```
# create one hot encoded vectors
image_vectors = [[0 for _ in range(len(all_labels))] for _ in␣
 ↪range(len(image_to_labels))]

# map label to an index
label_to_index = dict(map(lambda x: (x[1], x[0]), enumerate(all_labels)))

# populate vectors
for i, (image, labels) in enumerate(image_to_labels.items()):
    image_vector = image_vectors[i]
    for label in labels:
        image_vector[label_to_index[label]] = 1
```

Affinity Propagation failed to converge!

[33]:
```
# !pip3 install scikit-learn
# from sklearn.cluster import AffinityPropagation
# af = AffinityPropagation()
# af = af.fit(image_vectors)
```

[34]:
```
from sklearn.cluster import FeatureAgglomeration
```

Feature Agglomeration doesn't give us good clusters, also, there is ZERO EXPLAINABILITY

[35]:
```
labels_clusterings = []
```

[36]:
```
for n_clusters in range(2, 15):
    f = FeatureAgglomeration(affinity='manhattan',n_clusters=n_clusters,␣
 ↪linkage='complete')
    f.fit(image_vectors)
    labels_clusterings.append(f.labels_)
labels_clusterings[0:10]
```

[36]:
```
[array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 2, 1, 0]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 3, 0, 0, 0, 2, 1, 0]),
```

```
array([0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0, 0, 3, 0, 0, 0, 2, 1, 0]),
array([0, 6, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0, 0, 3, 0, 0, 0, 2, 1, 0]),
array([0, 6, 0, 0, 0, 5, 1, 0, 1, 0, 4, 0, 0, 7, 0, 0, 0, 2, 3, 0]),
array([0, 6, 0, 0, 8, 5, 1, 0, 1, 0, 4, 0, 0, 7, 0, 0, 0, 2, 3, 0]),
array([0, 6, 0, 0, 8, 5, 4, 1, 4, 0, 9, 0, 0, 7, 1, 0, 1, 2, 3, 0]),
array([ 0,  6,  0,  0,  8,  5,  4,  1,  4,  0,  9,  0,  0,  7,  1, 10,  1,
        2,  3,  0])]
```

[37]:
```
index_to_label = dict(enumerate(all_labels))
label_clusters = [{} for _ in range(len(labels_clusterings))]
for j, clustering in enumerate(labels_clusterings):
    dict_ = label_clusters[j]
    for i, l in enumerate(clustering):
        dict_[l] = dict_.get(l, []) + [index_to_label[i]]
```

[38]:
```
label_clusters
```

[38]:
```
[{0: ['bicycle',
   'bird',
   'boat',
   'sheep',
   'aeroplane',
   'cat',
   'diningtable',
   'sofa',
   'bottle',
   'bus',
   'chair',
   'motorbike',
   'cow',
   'tvmonitor',
   'train',
   'pottedplant',
   'dog',
   'car',
   'horse'],
  1: ['person']},
 {0: ['bicycle',
   'bird',
   'boat',
   'sheep',
   'aeroplane',
   'cat',
   'diningtable',
   'sofa',
   'bottle',
   'bus',
```

```
  'chair',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'car',
  'horse'],
 1: ['person'],
 2: ['dog']},
{0: ['bicycle',
  'bird',
  'boat',
  'sheep',
  'aeroplane',
  'cat',
  'diningtable',
  'sofa',
  'bottle',
  'bus',
  'chair',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'horse'],
 3: ['person'],
 2: ['dog'],
 1: ['car']},
{0: ['bicycle',
  'bird',
  'boat',
  'sheep',
  'aeroplane',
  'cat',
  'diningtable',
  'sofa',
  'bottle',
  'bus',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'horse'],
 4: ['chair'],
```

```
 3: ['person'],
 2: ['dog'],
 1: ['car']},
{0: ['bicycle',
  'bird',
  'boat',
  'sheep',
  'aeroplane',
  'diningtable',
  'sofa',
  'bottle',
  'bus',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'horse'],
 5: ['cat'],
 4: ['chair'],
 3: ['person'],
 2: ['dog'],
 1: ['car']},
{0: ['bicycle',
  'boat',
  'sheep',
  'aeroplane',
  'diningtable',
  'sofa',
  'bottle',
  'bus',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'horse'],
 6: ['bird'],
 5: ['cat'],
 4: ['chair'],
 3: ['person'],
 2: ['dog'],
 1: ['car']},
{0: ['bicycle',
  'boat',
  'sheep',
  'aeroplane',
```

```
  'sofa',
  'bus',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'horse'],
 6: ['bird'],
 5: ['cat'],
 1: ['diningtable', 'bottle'],
 4: ['chair'],
 7: ['person'],
 2: ['dog'],
 3: ['car']},
{0: ['bicycle',
  'boat',
  'sheep',
  'sofa',
  'bus',
  'motorbike',
  'cow',
  'tvmonitor',
  'train',
  'pottedplant',
  'horse'],
 6: ['bird'],
 8: ['aeroplane'],
 5: ['cat'],
 1: ['diningtable', 'bottle'],
 4: ['chair'],
 7: ['person'],
 2: ['dog'],
 3: ['car']},
{0: ['bicycle', 'boat', 'sheep', 'bus', 'motorbike', 'cow', 'train', 'horse'],
 6: ['bird'],
 8: ['aeroplane'],
 5: ['cat'],
 4: ['diningtable', 'bottle'],
 1: ['sofa', 'tvmonitor', 'pottedplant'],
 9: ['chair'],
 7: ['person'],
 2: ['dog'],
 3: ['car']},
{0: ['bicycle', 'boat', 'sheep', 'bus', 'motorbike', 'cow', 'horse'],
 6: ['bird'],
 8: ['aeroplane'],
```

```
 5: ['cat'],
 4: ['diningtable', 'bottle'],
 1: ['sofa', 'tvmonitor', 'pottedplant'],
 9: ['chair'],
 7: ['person'],
 10: ['train'],
 2: ['dog'],
 3: ['car']},
{11: ['bicycle'],
 6: ['bird'],
 0: ['boat', 'sheep', 'bus', 'motorbike', 'cow', 'horse'],
 8: ['aeroplane'],
 5: ['cat'],
 4: ['diningtable', 'bottle'],
 1: ['sofa', 'tvmonitor', 'pottedplant'],
 9: ['chair'],
 7: ['person'],
 10: ['train'],
 2: ['dog'],
 3: ['car']},
{11: ['bicycle'],
 6: ['bird'],
 1: ['boat', 'sheep', 'bus', 'cow', 'horse'],
 8: ['aeroplane'],
 12: ['cat'],
 4: ['diningtable', 'bottle'],
 0: ['sofa', 'tvmonitor', 'pottedplant'],
 9: ['chair'],
 5: ['motorbike'],
 7: ['person'],
 10: ['train'],
 2: ['dog'],
 3: ['car']},
{11: ['bicycle'],
 13: ['bird'],
 0: ['boat', 'sheep', 'bus', 'cow', 'horse'],
 8: ['aeroplane'],
 12: ['cat'],
 1: ['diningtable', 'bottle'],
 4: ['sofa', 'pottedplant'],
 9: ['chair'],
 5: ['motorbike'],
 7: ['person'],
 6: ['tvmonitor'],
 10: ['train'],
 2: ['dog'],
 3: ['car']}]
```

Ok, let's try DBSCAN, its significantly different from the previous two that we tried

[39]:
```
from sklearn.cluster import DBSCAN
db = DBSCAN()
db = db.fit(image_vectors)
```

[40]:
```
db.labels_
```

[40]:
```
array([  0,   1,   1, …, 118, 118, 118])
```

In DBSCAN, the label -1 is an outlier image(noisy data)

[41]:
```
noisy_images = []
```

[42]:
```
for i,label in enumerate(db.labels_):
    if label == -1:
        noisy_images.append(i)

image_label_list = list(image_to_labels.items())
noisy_images_ = list(map(lambda x: image_label_list[x][0], noisy_images))
```

[43]:
```
# feel free to look at some of the outliers, change the slice values
# display(*map(lambda x: display_image(x), noisy_images_[0:2]))
```

[44]:
```
for noisy_image in noisy_images:
    image, labels = image_label_list[noisy_image]
#     print(labels)
```

[45]:
```
# list(image_to_labels).index(get_images_with_label_pair("aeroplane", "chair"))
```

[46]:
```
# in each cluster there are image indices that index in to the list returning␣
↪the labels. grab those labels!
```

[47]:
```
label_frequency_in_clusters = {}
for index, cluster_id in enumerate(db.labels_):
    labels = image_label_list[index][1]

    cluster = label_frequency_in_clusters.get(cluster_id,{})
    for label in labels:
        cluster[label] = cluster.get(label, 0) + 1

    label_frequency_in_clusters[cluster_id] = cluster
```

Some of the clusters provided by DBSCAN

[48]:
```
list(label_frequency_in_clusters.items())[10:20]
```

[48]:
```
[(9, {'car': 53, 'bus': 53, 'person': 53}),
 (10, {'car': 41, 'person': 41, 'bicycle': 41}),
 (11, {'car': 7, 'person': 7, 'boat': 7}),
 (12, {'car': 6, 'bottle': 6, 'person': 6}),
 (13, {'car': 15, 'train': 15}),
 (14, {'car': 7, 'pottedplant': 7}),
 (15, {'car': 30, 'motorbike': 30}),
 (16, {'car': 72, 'bus': 72}),
 (17, {'car': 16, 'dog': 16}),
 (18, {'car': 12, 'person': 12, 'dog': 12})]
```

That's weird, why are all the frequencies same?
Turns out that a lot of clusters have a high count and then it falls down significantly for the other label groups

We need a generic approach that can help in explaining the clusters too!

2.4.2 Let's build a graph!

The nodes in our graph are label clusters And there is a directed edge from node A to node B if node A is a maximal subset of node b

[49]:
```
nodes = dict(label_frequency_in_clusters)
del nodes[-1]
nodes_ = {}
for k, v in nodes.items():
    nodes_[k] = set(v.keys())

nodes = nodes_
```

[50]:
```
start = len(nodes)

new_unique_labels = set()
for index, cluster_id in enumerate(db.labels_):
    if cluster_id == -1:
        labels = image_label_list[index][1]
        new_unique_labels.add(frozenset(labels))
for s in list(map(lambda x: set(x), new_unique_labels)):
    nodes[start] = s
    start+=1
```

[51]:
```
nodes
```

[51]:
```
{0: {'horse', 'person'},
 1: {'horse'},
 2: {'car', 'horse', 'person'},
 3: {'car', 'horse'},
```

```
4: {'car'},
5: {'car', 'person'},
6: {'car', 'motorbike', 'person'},
7: {'bicycle', 'car'},
8: {'aeroplane', 'car'},
9: {'bus', 'car', 'person'},
10: {'bicycle', 'car', 'person'},
11: {'boat', 'car', 'person'},
12: {'bottle', 'car', 'person'},
13: {'car', 'train'},
14: {'car', 'pottedplant'},
15: {'car', 'motorbike'},
16: {'bus', 'car'},
17: {'car', 'dog'},
18: {'car', 'dog', 'person'},
19: {'aeroplane', 'car', 'person'},
20: {'boat', 'car'},
21: {'cat'},
22: {'cat', 'person'},
23: {'bottle', 'cat'},
24: {'cat', 'chair'},
25: {'cat', 'chair', 'diningtable'},
26: {'cat', 'dog'},
27: {'cat', 'chair', 'person'},
28: {'cat', 'pottedplant'},
29: {'cat', 'tvmonitor'},
30: {'cat', 'sofa'},
31: {'tvmonitor'},
32: {'bottle', 'person', 'tvmonitor'},
33: {'chair', 'tvmonitor'},
34: {'chair', 'sofa', 'tvmonitor'},
35: {'bottle', 'tvmonitor'},
36: {'bottle', 'chair', 'person', 'tvmonitor'},
37: {'person', 'tvmonitor'},
38: {'bottle', 'chair', 'tvmonitor'},
39: {'chair', 'person', 'tvmonitor'},
40: {'pottedplant', 'tvmonitor'},
41: {'sofa', 'tvmonitor'},
42: {'person', 'sofa', 'tvmonitor'},
43: {'chair', 'pottedplant', 'tvmonitor'},
44: {'chair', 'diningtable', 'person', 'tvmonitor'},
45: {'chair', 'person', 'pottedplant', 'tvmonitor'},
46: {'person', 'pottedplant', 'tvmonitor'},
47: {'pottedplant', 'sofa', 'tvmonitor'},
48: {'bird'},
49: {'bird', 'person'},
50: {'bird', 'boat'},
```

```
51: {'bottle'},
52: {'bottle', 'person'},
53: {'bottle', 'chair', 'person'},
54: {'bottle', 'person', 'sofa'},
55: {'bottle', 'diningtable', 'person'},
56: {'bottle', 'diningtable'},
57: {'bottle', 'chair', 'diningtable'},
58: {'bottle', 'dog'},
59: {'bottle', 'chair', 'diningtable', 'person'},
60: {'bottle', 'chair'},
61: {'bottle', 'pottedplant'},
62: {'bottle', 'dog', 'person'},
63: {'bottle', 'chair', 'diningtable', 'pottedplant'},
64: {'bicycle', 'bottle'},
65: {'bicycle', 'bottle', 'person'},
66: {'bottle', 'chair', 'diningtable', 'person', 'pottedplant'},
67: {'bottle', 'diningtable', 'person', 'pottedplant'},
68: {'aeroplane'},
69: {'aeroplane', 'person'},
70: {'pottedplant'},
71: {'chair', 'pottedplant'},
72: {'person', 'pottedplant'},
73: {'pottedplant', 'sofa'},
74: {'person', 'pottedplant', 'train'},
75: {'chair', 'pottedplant', 'sofa'},
76: {'chair', 'person', 'pottedplant'},
77: {'chair', 'diningtable', 'person', 'pottedplant'},
78: {'dog', 'person', 'pottedplant'},
79: {'dog', 'pottedplant'},
80: {'motorbike', 'pottedplant'},
81: {'chair', 'diningtable', 'pottedplant', 'sofa'},
82: {'chair', 'diningtable', 'pottedplant'},
83: {'person', 'pottedplant', 'sofa'},
84: {'bicycle', 'pottedplant'},
85: {'person', 'train'},
86: {'train'},
87: {'chair', 'diningtable', 'person'},
88: {'chair', 'diningtable'},
89: {'diningtable', 'person'},
90: {'diningtable'},
91: {'chair', 'diningtable', 'sofa'},
92: {'person', 'sofa'},
93: {'chair', 'sofa'},
94: {'dog', 'person', 'sofa'},
95: {'dog', 'sofa'},
96: {'sofa'},
97: {'chair', 'person', 'sofa'},
```

```
98: {'motorbike', 'person'},
99: {'motorbike'},
100: {'bus', 'person'},
101: {'bus'},
102: {'dog', 'person'},
103: {'chair', 'person'},
104: {'bicycle', 'person'},
105: {'boat', 'person'},
106: {'person'},
107: {'chair', 'dog', 'person'},
108: {'person', 'sheep'},
109: {'cow', 'person'},
110: {'boat'},
111: {'dog'},
112: {'dog', 'sheep'},
113: {'chair', 'dog'},
114: {'cow'},
115: {'bicycle'},
116: {'bicycle', 'chair'},
117: {'chair'},
118: {'sheep'},
119: {'diningtable', 'pottedplant', 'sofa', 'tvmonitor'},
120: {'pottedplant', 'train'},
121: {'bottle', 'pottedplant', 'tvmonitor'},
122: {'chair', 'dog', 'person', 'pottedplant'},
123: {'car', 'chair', 'motorbike', 'person'},
124: {'chair', 'diningtable', 'dog', 'pottedplant'},
125: {'bicycle', 'motorbike'},
126: {'bicycle', 'bus', 'person'},
127: {'bottle', 'sofa'},
128: {'bottle', 'chair', 'dog'},
129: {'dog', 'person', 'tvmonitor'},
130: {'bicycle', 'car', 'motorbike'},
131: {'bird', 'bottle', 'chair'},
132: {'chair', 'diningtable', 'tvmonitor'},
133: {'bicycle', 'chair', 'diningtable'},
134: {'bicycle', 'chair', 'person'},
135: {'aeroplane', 'boat'},
136: {'cat', 'diningtable'},
137: {'bicycle', 'car', 'chair', 'motorbike', 'person'},
138: {'bus', 'car', 'motorbike'},
139: {'car', 'dog', 'motorbike', 'person'},
140: {'bicycle', 'train'},
141: {'car', 'chair', 'dog'},
142: {'bottle', 'cat', 'tvmonitor'},
143: {'bicycle', 'chair', 'diningtable', 'person', 'pottedplant'},
144: {'bicycle', 'cow', 'person'},
```

```
145: {'boat', 'car', 'motorbike'},
146: {'car', 'motorbike', 'person', 'pottedplant'},
147: {'bottle', 'diningtable', 'person', 'sofa'},
148: {'chair', 'train'},
149: {'boat', 'chair'},
150: {'bird', 'sheep'},
151: {'bottle', 'person', 'sofa', 'tvmonitor'},
152: {'bottle', 'motorbike', 'person'},
153: {'chair', 'horse'},
154: {'bird', 'horse'},
155: {'bottle', 'car', 'diningtable', 'person'},
156: {'horse', 'person', 'sheep'},
157: {'cat', 'chair', 'tvmonitor'},
158: {'boat', 'bottle', 'chair', 'diningtable', 'person'},
159: {'cat', 'person', 'sofa'},
160: {'bottle', 'car', 'dog', 'person'},
161: {'bicycle', 'chair', 'tvmonitor'},
162: {'bird', 'person', 'pottedplant'},
163: {'cow', 'horse'},
164: {'cat', 'person', 'pottedplant'},
165: {'chair', 'diningtable', 'dog', 'sofa'},
166: {'bus', 'train'},
167: {'chair', 'dog', 'person', 'sofa'},
168: {'bus', 'person', 'train'},
169: {'bicycle', 'bus', 'car', 'person'},
170: {'boat', 'bus', 'person'},
171: {'bicycle', 'person', 'sofa', 'tvmonitor'},
172: {'chair', 'dog', 'person', 'sofa', 'tvmonitor'},
173: {'bicycle', 'motorbike', 'person'},
174: {'cat', 'sofa', 'tvmonitor'},
175: {'bird', 'pottedplant'},
176: {'cat', 'chair', 'person', 'pottedplant', 'sofa'},
177: {'bicycle', 'car', 'person', 'pottedplant'},
178: {'bicycle', 'bottle', 'motorbike'},
179: {'bottle', 'person', 'pottedplant', 'sofa'},
180: {'dog', 'person', 'sheep'},
181: {'boat', 'dog', 'person'},
182: {'bottle', 'diningtable', 'person', 'pottedplant', 'sofa'},
183: {'bird', 'chair', 'person'},
184: {'bird', 'person', 'sheep'},
185: {'aeroplane', 'car', 'chair', 'person'},
186: {'bicycle', 'cat', 'chair'},
187: {'bird', 'boat', 'person'},
188: {'bird', 'chair'},
189: {'bottle', 'motorbike', 'person', 'sheep'},
190: {'bottle', 'dog', 'tvmonitor'},
191: {'dog', 'person', 'sofa', 'tvmonitor'},
```

```
192: {'bottle', 'chair', 'sofa'},
193: {'bicycle', 'bus', 'pottedplant'},
194: {'bottle', 'chair', 'dog', 'person'},
195: {'bottle', 'cat', 'pottedplant'},
196: {'bicycle', 'person', 'pottedplant'},
197: {'chair', 'person', 'train'},
198: {'car', 'diningtable', 'person'},
199: {'bus', 'motorbike', 'sheep'},
200: {'bottle', 'horse', 'person'},
201: {'horse', 'motorbike'},
202: {'bicycle', 'bottle', 'person', 'sofa'},
203: {'car', 'person', 'train'},
204: {'bicycle', 'boat', 'person'},
205: {'car', 'chair'},
206: {'aeroplane', 'bird'},
207: {'car', 'person', 'sheep'},
208: {'bicycle', 'bird'},
209: {'bird', 'tvmonitor'},
210: {'cat', 'pottedplant', 'sofa'},
211: {'bicycle', 'cat'},
212: {'chair', 'dog', 'person', 'pottedplant', 'sofa'},
213: {'bird', 'cow'},
214: {'chair', 'dog', 'person', 'tvmonitor'},
215: {'dog', 'tvmonitor'},
216: {'person', 'sofa', 'train'},
217: {'bus', 'dog', 'person'},
218: {'bottle', 'car'},
219: {'bottle', 'chair', 'diningtable', 'person', 'tvmonitor'},
220: {'bicycle', 'chair', 'sofa'},
221: {'bottle', 'person', 'pottedplant'},
222: {'cat', 'diningtable', 'tvmonitor'},
223: {'cat', 'person', 'tvmonitor'},
224: {'bicycle', 'cat', 'pottedplant'},
225: {'bottle', 'diningtable', 'pottedplant'},
226: {'bus', 'car', 'motorbike', 'person'},
227: {'cat', 'chair', 'sofa'},
228: {'bottle', 'cat', 'chair', 'diningtable'},
229: {'bird', 'chair', 'dog', 'pottedplant'},
230: {'chair', 'diningtable', 'person', 'pottedplant', 'tvmonitor'},
231: {'boat', 'person', 'train'},
232: {'cat', 'chair', 'pottedplant'},
233: {'bicycle', 'bottle', 'car', 'person'},
234: {'boat', 'bottle', 'person'},
235: {'boat', 'motorbike'},
236: {'chair', 'dog', 'tvmonitor'},
237: {'bird', 'bottle'},
238: {'bottle', 'diningtable', 'person', 'tvmonitor'},
```

```
239: {'bicycle', 'bottle', 'person', 'pottedplant'},
240: {'diningtable', 'sofa'},
241: {'chair', 'diningtable', 'person', 'sofa'},
242: {'bird', 'cat'},
243: {'car', 'cow'},
244: {'car', 'person', 'sofa'},
245: {'dog', 'horse', 'person'},
246: {'bottle', 'chair', 'person', 'pottedplant'},
247: {'motorbike', 'person', 'pottedplant'},
248: {'bicycle', 'dog', 'person'},
249: {'bottle', 'cat', 'person', 'sofa'},
250: {'bottle', 'dog', 'person', 'sofa', 'tvmonitor'},
251: {'horse', 'person', 'pottedplant'},
252: {'boat', 'bus'},
253: {'car', 'cat'},
254: {'bottle', 'chair', 'sofa', 'tvmonitor'},
255: {'car', 'motorbike', 'pottedplant'},
256: {'cow', 'dog'},
257: {'car', 'cat', 'dog', 'person'},
258: {'car', 'chair', 'diningtable'},
259: {'car', 'chair', 'person'},
260: {'boat', 'chair', 'pottedplant', 'tvmonitor'},
261: {'bicycle', 'motorbike', 'sofa'},
262: {'dog', 'pottedplant', 'sofa'},
263: {'chair', 'diningtable', 'person', 'sofa', 'tvmonitor'},
264: {'chair', 'pottedplant', 'sofa', 'tvmonitor'},
265: {'bicycle', 'bus'},
266: {'dog', 'horse'},
267: {'car', 'person', 'tvmonitor'},
268: {'bottle', 'horse'},
269: {'bicycle', 'car', 'pottedplant'},
270: {'car', 'chair', 'diningtable', 'person'},
271: {'car', 'cow', 'person'},
272: {'aeroplane', 'pottedplant'},
273: {'bottle', 'diningtable', 'dog'},
274: {'bicycle', 'chair', 'person', 'pottedplant'},
275: {'dog', 'motorbike'},
276: {'bicycle', 'bus', 'car'},
277: {'car', 'chair', 'diningtable', 'pottedplant'},
278: {'aeroplane', 'bus', 'car'},
279: {'boat', 'chair', 'motorbike', 'person', 'pottedplant'},
280: {'chair', 'motorbike', 'person'},
281: {'boat', 'chair', 'person'},
282: {'car', 'person', 'pottedplant'},
283: {'dog', 'motorbike', 'person'},
284: {'cat', 'dog', 'sofa'},
285: {'chair', 'person', 'pottedplant', 'sofa'},
```

```
286: {'car', 'tvmonitor'},
287: {'car', 'chair', 'person', 'tvmonitor'},
288: {'bicycle', 'tvmonitor'},
289: {'bottle', 'chair', 'dog', 'person', 'sofa'},
290: {'diningtable', 'pottedplant'},
291: {'bicycle', 'person', 'train'},
292: {'chair', 'horse', 'person', 'tvmonitor'},
293: {'bottle', 'chair', 'diningtable', 'person', 'sofa'},
294: {'bird', 'car'},
295: {'diningtable', 'dog'},
296: {'bottle', 'chair', 'person', 'sofa'},
297: {'bottle', 'chair', 'diningtable', 'sofa'},
298: {'bus', 'motorbike', 'person'},
299: {'bottle', 'cat', 'diningtable', 'pottedplant'},
300: {'boat', 'chair', 'diningtable', 'person', 'pottedplant'},
301: {'boat', 'chair', 'diningtable', 'person'},
302: {'bird', 'diningtable'},
303: {'boat', 'cow'},
304: {'cow', 'sheep'},
305: {'cat', 'pottedplant', 'tvmonitor'},
306: {'cat', 'chair', 'diningtable', 'pottedplant'},
307: {'bottle', 'chair', 'diningtable', 'tvmonitor'},
308: {'bottle', 'bus', 'person'},
309: {'diningtable', 'dog', 'person'},
310: {'car', 'sheep'},
311: {'bottle', 'car', 'chair', 'person'},
312: {'cat', 'cow', 'person'},
313: {'boat', 'pottedplant', 'sofa'},
314: {'car', 'chair', 'dog', 'person'},
315: {'bottle', 'chair', 'person', 'sofa', 'tvmonitor'},
316: {'bicycle', 'dog'},
317: {'horse', 'train'},
318: {'bicycle', 'bottle', 'person', 'tvmonitor'},
319: {'cat', 'sheep'},
320: {'chair', 'diningtable', 'pottedplant', 'tvmonitor'},
321: {'bicycle', 'chair', 'diningtable', 'person'},
322: {'diningtable', 'person', 'sofa'},
323: {'bird', 'dog'},
324: {'chair', 'person', 'sofa', 'tvmonitor'},
325: {'chair', 'diningtable', 'sofa', 'tvmonitor'},
326: {'bottle', 'person', 'sheep'},
327: {'aeroplane', 'car', 'motorbike', 'person'},
328: {'cat', 'dog', 'pottedplant'},
329: {'boat', 'motorbike', 'person'},
330: {'motorbike', 'person', 'sheep'},
331: {'boat', 'chair', 'sofa'},
332: {'bottle', 'person', 'pottedplant', 'tvmonitor'},
```

```
 333: {'bicycle', 'bus', 'car', 'motorbike', 'person'},
 334: {'car', 'cow', 'motorbike', 'person'},
 335: {'chair', 'dog', 'pottedplant'},
 336: {'bus', 'car', 'person', 'pottedplant', 'train'},
 337: {'boat', 'bus', 'car', 'person'},
 338: {'boat', 'pottedplant'}}
```

[52]:
```
label_set_counts = dict(label_frequency_in_clusters)
del label_set_counts[-1]
label_set_counts = list(label_set_counts.values())
label_set_counts = dict(map(lambda x: (frozenset(x.keys()), list(x.
↪values())[0]) ,label_set_counts))


for index, cluster_id in enumerate(db.labels_):
    if cluster_id == -1:
        labels = frozenset(image_label_list[index][1])
        label_set_counts[labels] = label_set_counts.get(labels, 0) + 1
```

[53]:
```
len(label_set_counts)
```

[53]: 339

[54]:
```
# get count of images with given labels

import functools

@functools.lru_cache(maxsize=None)
def get_count_given_labels(labels):
    count = 0
    for check, c in label_set_counts.items():
        if labels.issubset(check):
            count += c
    return count

@functools.lru_cache(maxsize=None)
def get_images_given_labels(labels):
    images = []
    for image, image_labels in image_to_labels.items():
        if labels.issubset(frozenset(image_labels)):
            images.append(image)
    return images
```

[55]:
```
from itertools import combinations
```

[56]:
```
possible_edges = list(combinations(nodes,2))
```

[57]:
```
len(db.labels_)
```

[57]:
```
11540
```

[58]:
```
edge_list = []
for a, b in possible_edges:
    set_a = nodes[a]
    set_b = nodes[b]

    if len(set_a) > len(set_b):
        if len(set_a - set_b) == 1:
            edge_list.append((b,a, {"weight":
 ↪get_count_given_labels(frozenset(set_a))/
 ↪get_count_given_labels(frozenset(set_b))}))

        continue

    elif len(set_a) < len(set_b):
        if len(set_b - set_a) == 1:
            edge_list.append((a,b, {"weight":␣
 ↪get_count_given_labels(frozenset(set_b))/
 ↪get_count_given_labels(frozenset(set_a))}))
```

[59]:
```
# !pip3 install networkx
# !pip3 install matplotlib
#!pip3 install ipycytoscape

import networkx as nx
import matplotlib

# import ipycytoscape
# import ipywidgets as widgets
# cytoscapeobj = ipycytoscape.CytoscapeWidget()

# deleted the cytoscape visualisation
# it wasn't good
```

[60]:
```
g = nx.DiGraph(edge_list)
```

[61]:
```
# nx.draw(g)
```

[62]:
```
ts = nx.topological_sort(g)
```

[63]:
```
leaf_nodes = [node for node in g.nodes() if g.in_degree(node)!=0 and g.
 ↪out_degree(node)==0]
```

[64]:
```
outer_nodes = {}
for leaf_node in leaf_nodes:

    min_w = 100
    for edge in list(g.in_edges(leaf_node)):
        min_w = min(min_w, g.get_edge_data(*edge)['weight'])

    outer_nodes[leaf_node] = min_w
```

[65]:
```
list(map(lambda x: (nodes[x[0]], x[1], x[0]),sorted(list(outer_nodes.items()),␣
↪key=lambda x: x[1])))
```

[65]:
```
[({'cat', 'sheep'}, 0.000925925925925926, 319),
 ({'bird', 'tvmonitor'}, 0.00130718954248366, 209),
 ({'aeroplane', 'bird'}, 0.00130718954248366, 206),
 ({'bicycle', 'bird'}, 0.00130718954248366, 208),
 ({'bird', 'diningtable'}, 0.00130718954248366, 302),
 ({'aeroplane', 'pottedplant'}, 0.0014925373134328358, 272),
 ({'cow', 'dog'}, 0.0015552099533437014, 256),
 ({'bird', 'car'}, 0.0017226528854435831, 294),
 ({'chair', 'horse'}, 0.0017873100983020554, 153),
 ({'horse', 'train'}, 0.001838235294117647, 317),
 ({'bird', 'cat'}, 0.00185185185185185, 242),
 ({'horse', 'motorbike'}, 0.0019011406844106464, 201),
 ({'boat', 'cow'}, 0.001968503937007874, 303),
 ({'car', 'person', 'sheep'}, 0.0022471910112359553, 207),
 ({'chair', 'person', 'train'}, 0.002347417840375587, 197),
 ({'bottle', 'horse', 'person'}, 0.002551020408163265, 200),
 ({'bottle', 'bus', 'person'}, 0.002551020408163265, 308),
 ({'bird', 'horse'}, 0.00261437908496732, 154),
 ({'bird', 'cow'}, 0.00261437908496732, 213),
 ({'cow', 'sheep'}, 0.003076923076923077, 304),
 ({'bicycle', 'person', 'train'}, 0.003355704697986577, 291),
 ({'bicycle', 'dog', 'person'}, 0.003355704697986577, 248),
 ({'bicycle', 'boat', 'person'}, 0.003355704697986577, 204),
 ({'diningtable', 'dog', 'person'}, 0.003745318352059925, 309),
 ({'bird', 'dog'}, 0.0038880248833592537, 323),
 ({'bus', 'dog', 'person'}, 0.004098360655737705, 217),
 ({'dog', 'person', 'sheep'}, 0.004098360655737705, 180),
 ({'cow', 'horse'}, 0.004149377593360996, 163),
 ({'car', 'cat'}, 0.004306632213608958, 253),
 ({'car', 'person', 'sofa'}, 0.0044943820224719105, 244),
 ({'horse', 'person', 'sheep'}, 0.004651162790697674, 156),
 ({'bird', 'chair', 'person'}, 0.004694835680751174, 183),
 ({'person', 'sofa', 'train'}, 0.005291005291005291, 216),
 ({'aeroplane', 'boat'}, 0.005970149253731343, 135),
 ({'cat', 'chair', 'tvmonitor'}, 0.005988023952095809, 157),
```

({'bicycle', 'chair', 'tvmonitor'}, 0.005988023952095809, 161),
({'boat', 'person', 'train'}, 0.006329113924050633, 231),
({'bottle', 'diningtable', 'dog'}, 0.006578947368421052, 273),
({'bicycle', 'cow', 'person'}, 0.006711409395973154, 144),
({'bird', 'person', 'pottedplant'}, 0.006756756756756757, 162),
({'bicycle', 'chair', 'sofa'}, 0.006896551724137931, 220),
({'boat', 'chair', 'sofa'}, 0.006896551724137931, 331),
({'boat', 'bottle', 'person'}, 0.007653061224489796, 234),
({'car', 'chair', 'diningtable', 'person'}, 0.007936507936507936, 270),
({'dog', 'horse', 'person'}, 0.00819672131147541, 245),
({'bird', 'bottle', 'chair'}, 0.008547008547008548, 131),
({'car', 'person', 'train'}, 0.008988764044943821, 203),
({'horse', 'person', 'pottedplant'}, 0.009302325581395349, 251),
({'bottle', 'car', 'diningtable', 'person'}, 0.009433962264150943, 155),
({'boat', 'car', 'motorbike'}, 0.010101010101010102, 145),
({'boat', 'motorbike', 'person'}, 0.01020408163265306, 329),
({'bus', 'person', 'train'}, 0.011111111111111112, 168),
({'boat', 'dog', 'person'}, 0.012295081967213115, 181),
({'bird', 'boat', 'person'}, 0.012658227848101266, 187),
({'bottle', 'car', 'chair', 'person'}, 0.014084507042253521, 311),
({'aeroplane', 'bus', 'car'}, 0.014184397163120567, 278),
({'cat', 'sofa', 'tvmonitor'}, 0.014492753623188406, 174),
({'boat', 'pottedplant', 'sofa'}, 0.014492753623188406, 313),
({'cat', 'chair', 'diningtable', 'pottedplant'}, 0.014705882352941176, 306),
({'chair', 'diningtable', 'dog', 'pottedplant'}, 0.014705882352941176, 124),
({'car', 'chair', 'diningtable', 'pottedplant'}, 0.014705882352941176, 277),
({'boat', 'bus', 'car', 'person'}, 0.015625, 337),
({'cat', 'person', 'tvmonitor'}, 0.015706806282722512, 223),
({'car', 'dog', 'motorbike', 'person'}, 0.016129032258064516, 139),
({'car', 'motorbike', 'person', 'pottedplant'}, 0.016129032258064516, 146),
({'aeroplane', 'car', 'motorbike', 'person'}, 0.016129032258064516, 327),
({'car', 'cow', 'motorbike', 'person'}, 0.016129032258064516, 334),
({'bottle', 'cat', 'tvmonitor'}, 0.01639344262295082, 142),
({'cat', 'chair', 'person'}, 0.01643192488262911, 27),
({'cat', 'cow', 'person'}, 0.01694915254237288, 312),
({'car', 'chair', 'person', 'tvmonitor'}, 0.01694915254237288, 287),
({'chair', 'horse', 'person', 'tvmonitor'}, 0.01694915254237288, 292),
({'bottle', 'cat', 'chair', 'diningtable'}, 0.017543859649122806, 228),
({'bicycle', 'car', 'person', 'pottedplant'}, 0.018867924528301886, 177),
({'bicycle', 'cat', 'chair'}, 0.018867924528301886, 186),
({'cat', 'person', 'pottedplant'}, 0.02027027027027027, 164),
({'cat', 'pottedplant', 'tvmonitor'}, 0.02040816326530612, 305),
({'cat', 'chair', 'sofa'}, 0.020689655172413793, 227),
({'bird', 'person', 'sheep'}, 0.020833333333333332, 184),
({'bicycle', 'motorbike', 'person'}, 0.02348993288590604, 173),
({'cat', 'pottedplant', 'sofa'}, 0.028985507246376812, 210),
({'car', 'horse', 'person'}, 0.029213483146067417, 2),

```
({'cat', 'dog', 'pottedplant'}, 0.03225806451612903, 328),
({'bicycle', 'bottle', 'person', 'tvmonitor'}, 0.03225806451612903, 318),
({'bottle', 'person', 'pottedplant', 'tvmonitor'}, 0.03225806451612903, 332),
({'bicycle', 'bottle', 'motorbike'}, 0.03225806451612903, 178),
({'bottle', 'dog', 'tvmonitor'}, 0.03278688524590164, 190),
({'boat', 'bottle', 'chair', 'diningtable', 'person'},
 0.03333333333333333,
 158),
({'bottle', 'chair', 'diningtable', 'person', 'sofa'},
 0.03333333333333333,
 293),
({'chair', 'diningtable', 'dog', 'sofa'}, 0.03333333333333333, 165),
({'bicycle', 'bottle', 'person', 'sofa'}, 0.03571428571428571, 202),
({'bottle', 'cat', 'person', 'sofa'}, 0.03571428571428571, 249),
({'bicycle', 'cat', 'pottedplant'}, 0.037037037037037035, 224),
({'boat', 'chair', 'pottedplant', 'tvmonitor'}, 0.037037037037037035, 260),
({'person', 'pottedplant', 'train'}, 0.04054054054054054, 74),
({'dog', 'pottedplant', 'sofa'}, 0.043478260869565216, 262),
({'bicycle', 'bottle', 'person', 'pottedplant'}, 0.045454545454545456, 239),
({'cat', 'diningtable', 'tvmonitor'}, 0.047619047619047616, 222),
({'bottle', 'cat', 'diningtable', 'pottedplant'}, 0.047619047619047616, 299),
({'chair', 'diningtable', 'person', 'pottedplant', 'tvmonitor'}, 0.05, 230),
({'bicycle', 'chair', 'diningtable', 'person', 'pottedplant'}, 0.05, 143),
({'boat', 'chair', 'diningtable', 'person', 'pottedplant'}, 0.05, 300),
({'bicycle', 'bus', 'pottedplant'}, 0.05263157894736842, 193),
({'bicycle', 'person', 'sofa', 'tvmonitor'}, 0.058823529411764705, 171),
({'bicycle', 'car', 'motorbike'}, 0.06060606060606061, 130),
({'bottle', 'car', 'dog', 'person'}, 0.0625, 160),
({'car', 'cat', 'dog', 'person'}, 0.0625, 257),
({'car', 'chair', 'dog', 'person'}, 0.0625, 314),
({'bicycle', 'motorbike', 'sofa'}, 0.0625, 261),
({'cat', 'dog', 'sofa'}, 0.06666666666666667, 284),
({'chair', 'pottedplant', 'sofa', 'tvmonitor'}, 0.06666666666666667, 264),
({'bicycle', 'bottle', 'car', 'person'}, 0.07547169811320754, 233),
({'bottle', 'chair', 'person', 'sofa', 'tvmonitor'},
 0.08333333333333333,
 315),
({'chair', 'diningtable', 'pottedplant', 'sofa'}, 0.08823529411764706, 81),
({'bottle', 'chair', 'diningtable', 'person', 'tvmonitor'}, 0.1, 219),
({'aeroplane', 'car', 'chair', 'person'}, 0.125, 185),
({'diningtable', 'pottedplant', 'sofa', 'tvmonitor'}, 0.125, 119),
({'chair', 'dog', 'person', 'sofa', 'tvmonitor'}, 0.14285714285714285, 172),
({'bird', 'chair', 'dog', 'pottedplant'}, 0.14285714285714285, 229),
({'bottle', 'diningtable', 'person', 'pottedplant', 'sofa'},
 0.15384615384615385,
 182),
({'bottle', 'chair', 'diningtable', 'person', 'pottedplant'},
```

```
 0.16666666666666666,
 66),
({'chair', 'dog', 'person', 'pottedplant', 'sofa'}, 0.2, 212),
({'bottle', 'chair', 'dog', 'person', 'sofa'}, 0.2, 289),
({'chair', 'diningtable', 'person', 'sofa', 'tvmonitor'}, 0.25, 263),
({'bottle', 'dog', 'person', 'sofa', 'tvmonitor'}, 0.25, 250),
({'bottle', 'motorbike', 'person', 'sheep'}, 0.25, 189),
({'cat', 'chair', 'person', 'pottedplant', 'sofa'}, 0.25, 176),
({'bicycle', 'bus', 'car', 'motorbike', 'person'}, 0.3333333333333333, 333),
({'bicycle', 'car', 'chair', 'motorbike', 'person'}, 0.5, 137)]
```

[66]:
```
from functools import reduce

for node in g.nodes:
    out_edges = list(g.out_edges(node))
    sum_weights = reduce(lambda x,y: x + g.edges[y]['weight'],out_edges, 0.0)
    for edge in out_edges:
        g.edges[edge]['weight']/=sum_weights
```

[67]:
```
ts = nx.topological_sort(g)
node_values = {}
for node in ts:

    val = 0
    if g.in_degree(node) == 0:
        node_values[node] = 1
    else:
        for parent in g.in_edges(node):
            val += g.get_edge_data(*parent)['weight'] * node_values[parent[0]]
        node_values[node] = val

nx.set_node_attributes(g,node_values, name="prob")
```

[68]:
```
sorted(map(lambda x: ( g.nodes[x]['prob'],nodes[x]), outer_nodes))
```

[68]:
```
[(0.0047157610143825504, {'bicycle', 'motorbike', 'sofa'}),
 (0.006255701362550691, {'bicycle', 'chair', 'sofa'}),
 (0.006336249871877301, {'horse', 'motorbike'}),
 (0.008586424036723262, {'bicycle', 'bus', 'pottedplant'}),
 (0.00864166862065595, {'bottle', 'bus', 'person'}),
 (0.009044973582810296, {'chair', 'horse', 'person', 'tvmonitor'}),
 (0.009270030274534163, {'bicycle', 'bottle', 'motorbike'}),
 (0.009347052905903694, {'chair', 'horse'}),
 (0.009554319339757533, {'car', 'person', 'sofa'}),
 (0.009634978780289648, {'boat', 'chair', 'sofa'}),
 (0.010151033582181487, {'horse', 'train'}),
 (0.010725329853446946, {'chair', 'diningtable', 'dog', 'sofa'}),
```

```
(0.010827929752398536, {'bicycle', 'person', 'sofa', 'tvmonitor'}),
(0.011270197228451502, {'aeroplane', 'pottedplant'}),
(0.01285708167662475, {'bus', 'dog', 'person'}),
(0.013878788592297667, {'bottle', 'diningtable', 'dog'}),
(0.015064905755670004, {'diningtable', 'dog', 'person'}),
(0.016041444562899788, {'bird', 'diningtable'}),
(0.0166435861927476, {'bird', 'tvmonitor'}),
(0.01696836484729068, {'boat', 'chair', 'pottedplant', 'tvmonitor'}),
(0.017026213470462813, {'bicycle', 'bird'}),
(0.017210639983488096, {'car', 'chair', 'diningtable', 'pottedplant'}),
(0.01878637699533222, {'cat', 'sheep'}),
(0.020668914025269727, {'bicycle', 'dog', 'person'}),
(0.021613349818891056, {'cat', 'diningtable', 'tvmonitor'}),
(0.021617782865554547, {'bicycle', 'chair', 'tvmonitor'}),
(0.021738195481640743, {'bird', 'chair', 'dog', 'pottedplant'}),
(0.02206108081934744, {'bicycle', 'cat', 'chair'}),
(0.022309923399061377, {'boat', 'car', 'motorbike'}),
(0.022673642854015737, {'bottle', 'dog', 'tvmonitor'}),
(0.022989510489510485, {'boat', 'cow'}),
(0.02340980718196106, {'bottle', 'car', 'chair', 'person'}),
(0.023745346658060128, {'bicycle', 'cat', 'pottedplant'}),
(0.02457372320386019, {'car', 'cat'}),
(0.02472929470295581, {'aeroplane', 'bird'}),
(0.026261253299215255, {'car', 'chair', 'diningtable', 'person'}),
(0.026290137716967175, {'cat', 'chair', 'tvmonitor'}),
(0.02735534649485431, {'cat', 'pottedplant', 'tvmonitor'}),
(0.028489965386617078, {'bottle', 'car', 'diningtable', 'person'}),
(0.029171910451720174, {'diningtable', 'pottedplant', 'sofa', 'tvmonitor'}),
(0.03004483032178474, {'cat', 'sofa', 'tvmonitor'}),
(0.0315229730058213, {'chair', 'diningtable', 'dog', 'pottedplant'}),
(0.03195822782819307, {'bird', 'car'}),
(0.033107191316146534, {'cow', 'sheep'}),
(0.03426650572525545, {'bicycle', 'boat', 'person'}),
(0.03757275399066444, {'bird', 'cat'}),
(0.03788287478271294, {'bird', 'horse'}),
(0.040232108317214695, {'cow', 'dog'}),
(0.04132812057876194, {'boat', 'pottedplant', 'sofa'}),
(0.043805295975763256, {'car', 'chair', 'person', 'tvmonitor'}),
(0.044267466844629594, {'bottle', 'cat', 'tvmonitor'}),
(0.04439576487769258, {'cow', 'horse'}),
(0.04555337179326411, {'bottle', 'person', 'pottedplant', 'tvmonitor'}),
(0.04905793090419034, {'cat', 'dog', 'pottedplant'}),
(0.049076894235200605, {'cat', 'pottedplant', 'sofa'}),
(0.04928311085930893, {'bicycle', 'bottle', 'person', 'sofa'}),
(0.04949485690697614, {'bicycle', 'bottle', 'person', 'tvmonitor'}),
(0.04969138670170439, {'dog', 'pottedplant', 'sofa'}),
(0.05415368483649976, {'person', 'sofa', 'train'}),
```

```
(0.054539147663957406, {'car', 'cat', 'dog', 'person'}),
(0.05844645550527905, {'aeroplane', 'boat'}),
(0.06044769480876045, {'bottle', 'horse', 'person'}),
(0.06617036299990325, {'chair', 'person', 'train'}),
(0.06621438263229307, {'bird', 'cow'}),
(0.06795216334789618, {'bicycle', 'bottle', 'person', 'pottedplant'}),
(0.06847156975541616, {'cat', 'chair', 'sofa'}),
(0.07372697794195235, {'bicycle', 'car', 'motorbike'}),
(0.07469346935358437, {'bicycle', 'person', 'train'}),
(0.07508410249601948, {'cat', 'person', 'pottedplant'}),
(0.07542507872997391, {'bird', 'bottle', 'chair'}),
(0.0792010990214215, {'boat', 'person', 'train'}),
(0.08173676724128859, {'bottle', 'cat', 'diningtable', 'pottedplant'}),
(0.08429804555558765, {'bird', 'dog'}),
(0.08686077601723624, {'bottle', 'cat', 'chair', 'diningtable'}),
(0.08808779200159941, {'boat', 'bottle', 'person'}),
(0.09014706677029247, {'aeroplane', 'bus', 'car'}),
(0.09301515785171222, {'cat', 'person', 'tvmonitor'}),
(0.09821279478632555, {'cat', 'chair', 'person', 'pottedplant', 'sofa'}),
(0.09850835746714, {'car', 'chair', 'dog', 'person'}),
(0.10067392698594309, {'horse', 'person', 'pottedplant'}),
(0.10073403116950581, {'boat', 'dog', 'person'}),
(0.1075034200998885, {'bottle', 'car', 'dog', 'person'}),
(0.11989371722447113, {'bottle', 'cat', 'person', 'sofa'}),
(0.12131437545653817, {'horse', 'person', 'sheep'}),
(0.13253827275878713, {'dog', 'horse', 'person'}),
(0.13330353381052776, {'bird', 'person', 'pottedplant'}),
(0.13332503177056165, {'boat', 'motorbike', 'person'}),
(0.13487839314352743, {'bicycle', 'motorbike', 'person'}),
(0.13901912473013286, {'car', 'person', 'sheep'}),
(0.1426244040882026, {'cat', 'cow', 'person'}),
(0.1442409882368525, {'bus', 'person', 'train'}),
(0.15222616726027483, {'bicycle', 'car', 'person', 'pottedplant'}),
(0.15912358635189583,
 {'boat', 'chair', 'diningtable', 'person', 'pottedplant'}),
(0.16613357585081892, {'cat', 'chair', 'person'}),
(0.1668531389591747, {'chair', 'pottedplant', 'sofa', 'tvmonitor'}),
(0.17615442817916097, {'car', 'dog', 'motorbike', 'person'}),
(0.177836026741174, {'bottle', 'dog', 'person', 'sofa', 'tvmonitor'}),
(0.18558409706390583, {'bird', 'chair', 'person'}),
(0.18660737316196874, {'boat', 'bottle', 'chair', 'diningtable', 'person'}),
(0.19497906076592503, {'cat', 'chair', 'diningtable', 'pottedplant'}),
(0.2076018734068651, {'cat', 'dog', 'sofa'}),
(0.21835434400676174, {'bicycle', 'car', 'chair', 'motorbike', 'person'}),
(0.21980185883652908,
 {'bicycle', 'chair', 'diningtable', 'person', 'pottedplant'}),
(0.22574621394682456, {'car', 'motorbike', 'person', 'pottedplant'}),
```

```
 (0.23059258760883922, {'dog', 'person', 'sheep'}),
 (0.243347765155981, {'chair', 'diningtable', 'pottedplant', 'sofa'}),
 (0.2475371923064532,
  {'bottle', 'diningtable', 'person', 'pottedplant', 'sofa'}),
 (0.25811733567211165, {'bird', 'person', 'sheep'}),
 (0.27150532408197603, {'bicycle', 'cow', 'person'}),
 (0.2823730132907139, {'bottle', 'chair', 'diningtable', 'person', 'sofa'}),
 (0.3104531929446869, {'bottle', 'chair', 'dog', 'person', 'sofa'}),
 (0.3318275972001456, {'bottle', 'chair', 'person', 'sofa', 'tvmonitor'}),
 (0.34473703429324243, {'bird', 'boat', 'person'}),
 (0.35170516463572465, {'car', 'person', 'train'}),
 (0.37421940855564845, {'person', 'pottedplant', 'train'}),
 (0.3903201354638377, {'chair', 'dog', 'person', 'pottedplant', 'sofa'}),
 (0.40941273469186057, {'chair', 'dog', 'person', 'sofa', 'tvmonitor'}),
 (0.4396323723562249, {'bicycle', 'bottle', 'car', 'person'}),
 (0.4562764599026785,
  {'chair', 'diningtable', 'person', 'pottedplant', 'tvmonitor'}),
 (0.4735974663702541, {'aeroplane', 'car', 'chair', 'person'}),
 (0.4891286377721229, {'bottle', 'motorbike', 'person', 'sheep'}),
 (0.5324470243307791, {'boat', 'bus', 'car', 'person'}),
 (0.5489585607110854, {'aeroplane', 'car', 'motorbike', 'person'}),
 (0.5512700353244357, {'chair', 'diningtable', 'person', 'sofa', 'tvmonitor'}),
 (0.5712338300041314, {'car', 'cow', 'motorbike', 'person'}),
 (0.759392392633341, {'car', 'horse', 'person'}),
 (0.8333736168175704,
  {'bottle', 'chair', 'diningtable', 'person', 'tvmonitor'}),
 (1.2467146401069127,
  {'bottle', 'chair', 'diningtable', 'person', 'pottedplant'}),
 (2.088302664881321, {'bicycle', 'bus', 'car', 'motorbike', 'person'})]
```

[69]:
```
# display_image(get_images_given_labels(frozenset(['bottle', 'chair',␣
↪'diningtable', 'person', 'pottedplant', 'tvmonitor']))[0])
```

[70]:
```
# nodes
```

As we have false negatives in our graph, it makes sense to introduce partial labeling
Our previous graph only considers exact labelling, need to create a new graph

[71]:
```
attribute_sets_in_nodes = set()
for attrs in nodes.values():
    attribute_sets_in_nodes.add(frozenset(attrs))
```

[72]:
```
@functools.lru_cache(maxsize=None)
def check_attr_set(attr_set):
```

```
    count = get_count_given_labels(attr_set)
    if not attr_set in attribute_sets_in_nodes:
        return False, count
    else:
        return True, count
#     return 1
```

[73]:
```
attr_sets = {}

for attrs in attribute_sets_in_nodes:

    attrs_ = list(attrs)
    for r in range(1, len(attrs_)):
        combs = combinations(list(attrs_), r)
        for comb in combs:

            is_node, count = check_attr_set(frozenset(comb))
            attr_sets[frozenset(comb)] = count
```

[74]:
```
graph = nx.DiGraph()
```

[75]:
```
attr_sets_ = list(map(lambda x: (x[0], {"count":x[1]}), list(attr_sets.
↪items())))
```

[76]:
```
graph.add_nodes_from(attr_sets_)
names = dict([(x, ', '.join(list(x))) for x in graph.nodes()])
```

[77]:
```
possible_edges = list(combinations(list(attr_sets.items()), 2))
edges = []
for (a, a_count), (b, b_count) in possible_edges:
    if len(a) == len(b)+1:
        if b.issubset(a):
            edges.append((b,a, a_count/b_count))
    elif len(b) == len(a)+1:

        if a.issubset(b):
            edges.append((a,b,b_count/a_count))


graph.add_weighted_edges_from(edges)
```

[78]:
```
# edge with the min weights:
odds = list(sorted(edges, key=lambda x: x[2]))
```

You can check out some of the outlier images that we have

[79]:
```
# uncomment and run to view some images
# display(*list(map(lambda x: display_image(get_images_given_labels(x[1])[0]),␣
↪odds[0:10])))
```

[80]:
```
# move through the DAG and update the transitive node values and normalise the␣
↪edge weights
ts = nx.topological_sort(graph)
node_values = {}
for node in ts:

    val = 0
    if graph.in_degree(node) == 0:
        graph.nodes[node]['prob'] = 1
    else:
        for parent in graph.in_edges(node):
            val += graph.get_edge_data(*parent)['weight'] * graph.
↪nodes[parent[0]]['prob']
        graph.nodes[node]['prob'] = val

    out_edges = list(graph.out_edges(node))
    sum_weights = graph.nodes[node]['prob'] * reduce(lambda x,y: x + graph.
↪edges[y]['weight'],out_edges, 0.0)
    for child in out_edges:
        graph.edges[child]['weight'] /= sum_weights
```

[81]:
```
for node in graph.nodes:

    in_edges = list(graph.in_edges(node))
    sorted_in_edges = sorted(in_edges, key=lambda x: graph.edges[x]['weight'])
    if len(sorted_in_edges) > 1:

        # try different metrics for denoting oddness of a node

        # graph.nodes[node]['oddity'] = (graph.
↪edges[sorted_in_edges[-1]]['weight'] + graph.
↪edges[sorted_in_edges[-2]]['weight']) * graph.
↪edges[sorted_in_edges[0]]['weight']
        # graph.nodes[node]['oddity'] = reduce(lambda x,y: x+graph.
↪edges[y]['weight'], sorted_in_edges[0:2], 0.0)/len(sorted_in_edges)

        graph.nodes[node]['oddity'] = reduce(lambda x,y: x*graph.
↪edges[y]['weight'], sorted_in_edges, 1.0)

    elif len(sorted_in_edges) == 1:
        graph.nodes[node]['oddity'] = graph.edges[sorted_in_edges[0]]['weight']
```

```
    else:
        graph.nodes[node]['oddity'] = 1
```

[82]:
```
list(map(lambda x: (x, graph.nodes[x]['oddity']), sorted(list(graph.nodes),␣
↪key=lambda x: graph.nodes[x]['oddity'])))[0:10]
```

[82]:
```
[(frozenset({'bottle', 'bus'}), 3.1527441485068594e-06),
 (frozenset({'motorbike', 'sofa'}), 3.4202533723698255e-06),
 (frozenset({'bus', 'dog'}), 5.442887778539782e-06),
 (frozenset({'car', 'sofa'}), 6.503853533218432e-06),
 (frozenset({'aeroplane', 'chair'}), 6.5487884741322864e-06),
 (frozenset({'horse', 'tvmonitor'}), 6.66982371655917e-06),
 (frozenset({'boat', 'tvmonitor'}), 7.591919161244771e-06),
 (frozenset({'sofa', 'train'}), 9.4682623844872e-06),
 (frozenset({'bicycle', 'boat'}), 9.501639032733147e-06),
 (frozenset({'chair', 'horse'}), 1.0605325464181836e-05)]
```

2.5 Better results!

Here are some of the other weird/interesting images!

Sofa near a motorbike

[83]:
```
display_image(get_images_given_labels(frozenset({'motorbike', 'sofa'}))[0])
```

[83]:

A couple of images with cars and sofa together

[84]:
```
display_images_with_label_pair('car', 'sofa')
```

Again, its rare to see a horse and a screen in the same image!

[85]:
```
display_images_with_label_pair('horse', 'tvmonitor')
```

inaccurately labelled? the boat isn't a real boat

[86]:
```
display_images_with_label_pair('boat', 'tvmonitor')
```

this boat is real, but uhmm ..

[87]:
```
display_images_with_label_pair('boat', 'train')
```

Is this a real image?

[88]:
```
display_images_with_label_pair('sofa', 'train')
```

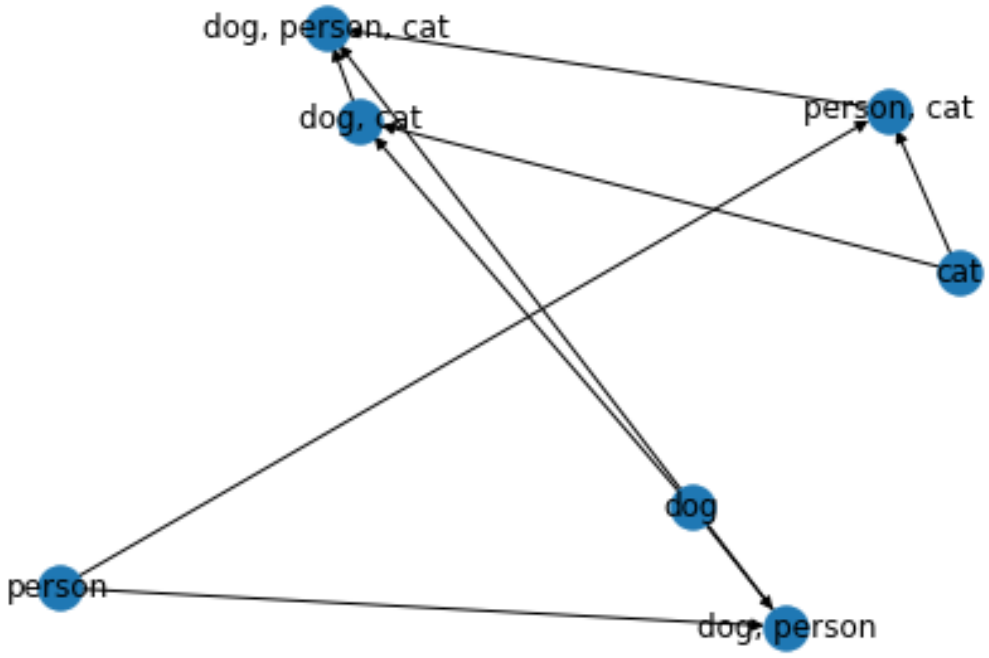
[89]:
```
# display(*list(map(lambda x:␣
 ↪display_image(get_images_given_labels(x[0][1])[0]), odds[0:100])))
```

2.5.1 To better explain how the graph was built:

[90]:
```
# lets look at some of the nodes of the graphs
# the subgraph contains all nodes related to dog, cat, and person
sub_nodes = [frozenset({"dog", "cat", "person"}),
 frozenset({"dog"}),
 frozenset({"cat"}),
 frozenset({"person"}),
 frozenset({"dog", "cat"}),
 frozenset({"dog", "person"}),
 frozenset({"cat", "person"}),
]

sub_names = dict(map(lambda x: (x, names[x]), sub_nodes))
subg = graph.subgraph([*sub_nodes])
```

[91]:
```
nx.draw(subg, labels=sub_names,)
```

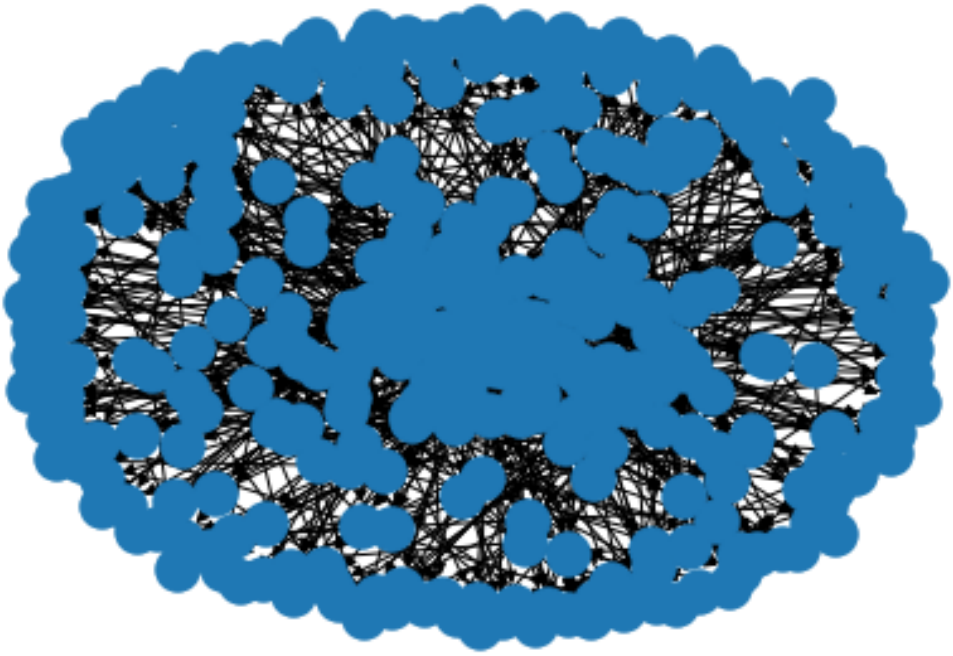


2.5.2 The edge directions should be obvious?

Each node and edge have weights and other attributes associated with them

2.5.3 So what happens if we draw the complete graph?

[92]:
```
nx.draw(graph)
```



2.5.4 Clearly we need a better visualisation library

[93]:
```
# !pip3 install plotly
from plotly.offline import download_plotlyjs, init_notebook_mode, iplot,plot
import plotly.graph_objs as go
init_notebook_mode(connected=True)
```

[94]:
```
pos = nx.spring_layout(graph, k=0.5, iterations=50)

for n, p in pos.items():
    graph.nodes[n]['pos'] = p


edge_trace = go.Scatter(
    x=[],
    y=[],
    line=dict(width=0.5,color='#888'),
    hoverinfo='none',
    mode='lines')


for edge in graph.edges():
```

```
    x0, y0 = graph.nodes[edge[0]]['pos']
    x1, y1 = graph.nodes[edge[1]]['pos']
    edge_trace['x'] += tuple([x0, x1, None])
    edge_trace['y'] += tuple([y0, y1, None])
node_trace = go.Scatter(
    x=[],
    y=[],
    text=[],
    mode='markers',
    hoverinfo='text',
    marker=dict(
        showscale=True,
        colorscale='reds',
        reversescale=True,
        color=[],
        size=15,
        colorbar=dict(
            thickness=10,
            title='Likelihood',
            xanchor='left',
            titleside='right'
        ),
        line=dict(width=0)))


for node in graph.nodes():
    x, y = graph.nodes[node]['pos']
    node_trace['x'] += tuple([x])
    node_trace['y'] += tuple([y])


node_trace['text'] = []
node_trace['marker']['color'] = []
for node in graph.nodes():
    node_trace['marker']['color']+=tuple([(min(1,graph.
 ↪nodes[node]['oddity']*1000)**2)])
    node_info = str(list(node))
    node_trace['text']+=tuple([node_info])

fig = go.Figure(data=[edge_trace, node_trace],
         layout=go.Layout(
            title='<br>Attribute Graph',
            titlefont=dict(size=16),
            showlegend=False,
            hovermode='closest',
            margin=dict(b=20,l=5,r=5,t=40),
            annotations=[ dict(
```

```
                text="",
                showarrow=False,
                xref="paper", yref="paper") ],
            xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
            yaxis=dict(showgrid=False, zeroline=False, showticklabels=False)))
```

[95]:
```
# inline plot
iplot(fig)

# uncomment for a separate window (bigger visualization)
# plot(fig)
```

2.5.5 Understanding the plot above:

- The likelihood is the inverse of oddity, a more likelihood node is lighter than a less likelihood node (the maroon ones are less likely label clusters)
- the threshold for coloring has been chosen to make the odd nodes stand out
- hover above a node to see the labels it corresponds to

P.N. - The graph was built to show oddity at the first point. This means that "tvmonitor", "ship" and "chair" has a low likelihood but that's because "tvmonitor" and "ship" are already unlikely. Thus to make it easier to understand the graph, the earliest point of oddness are highlighted

2.6 So, we have our graph

(stored in the variable 'graph')

- Each node in the graph denotes a set of labels
- Each edge denotes a new label that is added, so Node A ("dog") --> Node B ("dog", "cat")
- The weights of edges denote probability
- The nodes too have probabilities associated with them
- They also have the property - 'oddity' that denotes how odd an image with those set of labels are

2.7 What does this help us with?

We can: - **find odd images** > Find the 'odd' nodes, use the helper functions to get the images belonging in the cluster - **explain why an image is odd** > The image corresponds to a node, find the label with the lowest in-edges of the node, that label's presence makes the image odd, this can be extended to multiple labels - **find the common clusters** > Find the nodes with high 'prob' values - **predict a label that should be there in an image (and might be occluded)** > From the node for the image, find the highest probable paths (its a DAG) and the edges are weighted - **and so much more ...**

2.8 The tasks that we can perform/improve upon

- Anamoly Detection
- Data cleaning (incorrect labels prediction etc), data enriching
- Scene Understanding
- Captioning
- ...

[ ]:

[ ]: