Experiments

January 10, 2020

1 Predicting poverty with satellite imagery

Is it feasible to estimate the standard of living, energy consumption, or other socioeconomic metrics of a site based on satellite imagery?

For this project, we will focus on the country of Rwanda. We will need to download three distinct datasets, including DHS data, satellite data from the Google Maps API, as well as nighttime luminosity data. The DHS data requires registration (which can take several days to be approved), and the Google Maps API is rate-limited, so it will be slow. The deep learning section may also take several hours to compute.

1.1 Overview

These are the key steps in the problem set:

1. Download satellite night lights images from NOAA
2. Download DHS data for Rwanda
3. Test whether nightlights data can predict wealth, as observed in DHS
4. Download daytime satellite imagery from Google Maps
5. Test whether basic features of daytime imagery can predict wealth
6. Extract features from daytime imagery using deep learning libraries
7. Fine-tune existing CNN and incorporate transfer learning
8. Construct maps showing the predicted distribution of wealth in Rwanda

---

2 2. Download Rwandan DHS and construct cluster-level aggregates

- **INPUT**:
  - `rwanda_clusters_location.csv`: Coordinates of the centroid of each cluster
- **OUTPUT**:
  - `rwanda_cluster_avg_asset_2010.csv`: Comma-delimited file indicated average wealth of each cluster

Demographic and Health Surveys (DHS) are nationally-representative household surveys that provide data for a wide range of monitoring and impact evaluation indicators in the areas of population, health, and nutrition. For this, we will need to download the 2010 Rwandan DHS data, requesting for the GPS dataset.

[3]:
```
import wget

night_image_url = 'https://ngdc.noaa.gov/eog/data/web_data/v4composites/F182010.
↪v4.tar'
wget.download(night_image_url)
```

[3]: 'F182010.v4.tar'

3 2. Download Rwandan DHS and construct cluster-level aggregates

- **INPUT**:
  - `rwanda_clusters_location.csv`: Coordinates of the centroid of each cluster
- **OUTPUT**:
  - `rwanda_cluster_avg_asset_2010.csv`: Comma-delimited file indicated average wealth of each cluster

Demographic and Health Surveys (DHS) are nationally-representative household surveys that provide data for a wide range of monitoring and impact evaluation indicators in the areas of population, health, and nutrition. For this, we will need to download the 2010 Rwandan DHS data. Do not forget to request for the GPS dataset.

[ ]:
```
import pandas as pd
import numpy as np

file_name = 'data/DHS/RWHR61FL.DAT'
cluster_file = 'data/DHS/rwanda_clusters_location.csv'
cluster_all = []
wealth_all = []
with open(file_name) as f:
    for line in f:
        cluster = int(line[15:23])
        wealth = int(line[230:238]) / 100000.0
        cluster_all.append(cluster)
        wealth_all.append(wealth)

df = pd.DataFrame({'cluster': cluster_all, 'wlthindf': wealth_all})
cluster_avg_asset = df.groupby('cluster')['wlthindf'].median().reset_index()

df_location = pd.read_csv(cluster_file)[['DHSCLUST', 'LATNUM', 'LONGNUM']]
result = cluster_avg_asset.merge(df_location, how='inner', left_on='cluster',␣
↪right_on='DHSCLUST')[['cluster', 'wlthindf', 'LATNUM', 'LONGNUM']]
```

```
result.rename(columns={'LATNUM': 'latitude', 'LONGNUM':'longitude'},␣
 ↪inplace=True)
result.to_csv('intermediate_files/rwanda_cluster_avg_asset_2010.csv',␣
 ↪index=False)
```

4 3. Test whether night lights data can predict wealth, as observed in DHS

Now that we have "ground truth" measures of average cluster wealth, our goal is to understand whether the nightlights data can be used to predict wealth. First, we merge the DHS and nightlights data, and then fit a model of wealth on nightlights.

4.1 3.1 Merge nightlights and DHS data at cluster level

- **INPUT**:
- `F182010.v4d_web.stable_lights.avg_vis.tif`: Nightlights data, from Step 1
- `rwanda_cluster_avg_asset_2010.csv`: DHS cluster averages, from Step 2
- **OUTPUT**: Merged datasets
- `DHS_nightlights.csv`: Merged dataset with 492 rows, and 6 columns (one indicates average cluster wealth, 5 nightlights features)
- Scatterplot of nightlights vs. DHS wealth

Next, we erform a "spatial join" to compute the average nighttime luminosity for each of the DHS clusters. To do this, we should take the average of the luminosity values for the nightlights locations surrounding the cluster centroid. The output is saved as `DHS_nightlights.csv`.

[6]:
```
import time
import os
import os.path
from osgeo import gdal, ogr, osr
from scipy import ndimage
from scipy import misc
from io import StringIO
gdal.UseExceptions()
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import gridspec
%matplotlib inline
import urllib
import pandas as pd
import numpy as np

def read_raster(raster_file):
    """
    Function
```

```
    --------
    read_raster

    Given a raster file, get the pixel size, pixel location, and pixel value

    Parameters
    ----------
    raster_file : string
        Path to the raster file

    Returns
    -------
    x_size : float
        Pixel size
    top_left_x_coords : numpy.ndarray  shape: (number of columns,)
        Longitude of the top-left point in each pixel
    top_left_y_coords : numpy.ndarray  shape: (number of rows,)
        Latitude of the top-left point in each pixel
    centroid_x_coords : numpy.ndarray  shape: (number of columns,)
        Longitude of the centroid in each pixel
    centroid_y_coords : numpy.ndarray  shape: (number of rows,)
        Latitude of the centroid in each pixel
    bands_data : numpy.ndarray  shape: (number of rows, number of columns, 1)
        Pixel value
    """
    raster_dataset = gdal.Open(raster_file, gdal.GA_ReadOnly)
    # Get project coordination
    proj = raster_dataset.GetProjectionRef()
    bands_data = []
    # Loop through all raster bands
    for b in range(1, raster_dataset.RasterCount + 1):
        band = raster_dataset.GetRasterBand(b)
        bands_data.append(band.ReadAsArray())
        no_data_value = band.GetNoDataValue()
    bands_data = np.dstack(bands_data)
    rows, cols, n_bands = bands_data.shape

    # Get the metadata of the raster
    geo_transform = raster_dataset.GetGeoTransform()
    (upper_left_x, x_size, x_rotation, upper_left_y, y_rotation, y_size) =␣
↪geo_transform

    # Get location of each pixel
    x_size = 1.0 / int(round(1 / float(x_size)))
    y_size = - x_size
    y_index = np.arange(bands_data.shape[0])
    x_index = np.arange(bands_data.shape[1])
```

```python
    top_left_x_coords = upper_left_x + x_index * x_size
    top_left_y_coords = upper_left_y + y_index * y_size
    # Add half of the cell size to get the centroid of the cell
    centroid_x_coords = top_left_x_coords + (x_size / 2)
    centroid_y_coords = top_left_y_coords + (y_size / 2)

    return (x_size, top_left_x_coords, top_left_y_coords, centroid_x_coords,␣
 ↪centroid_y_coords, bands_data)

# Helper function to get the pixel index of the point
def get_cell_idx(lon, lat, top_left_x_coords, top_left_y_coords):
    """
    Function
    --------
    get_cell_idx

    Given a point location and all the pixel locations of the raster file,
    get the column and row index of the point in the raster

    Parameters
    ----------
    lon : float
        Longitude of the point
    lat : float
        Latitude of the point
    top_left_x_coords : numpy.ndarray  shape: (number of columns,)
        Longitude of the top-left point in each pixel
    top_left_y_coords : numpy.ndarray  shape: (number of rows,)
        Latitude of the top-left point in each pixel

    Returns
    -------
    lon_idx : int
        Column index
    lat_idx : int
        Row index
    """
    lon_idx = np.where(top_left_x_coords < lon)[0][-1]
    lat_idx = np.where(top_left_y_coords > lat)[0][-1]
    return lon_idx, lat_idx
```

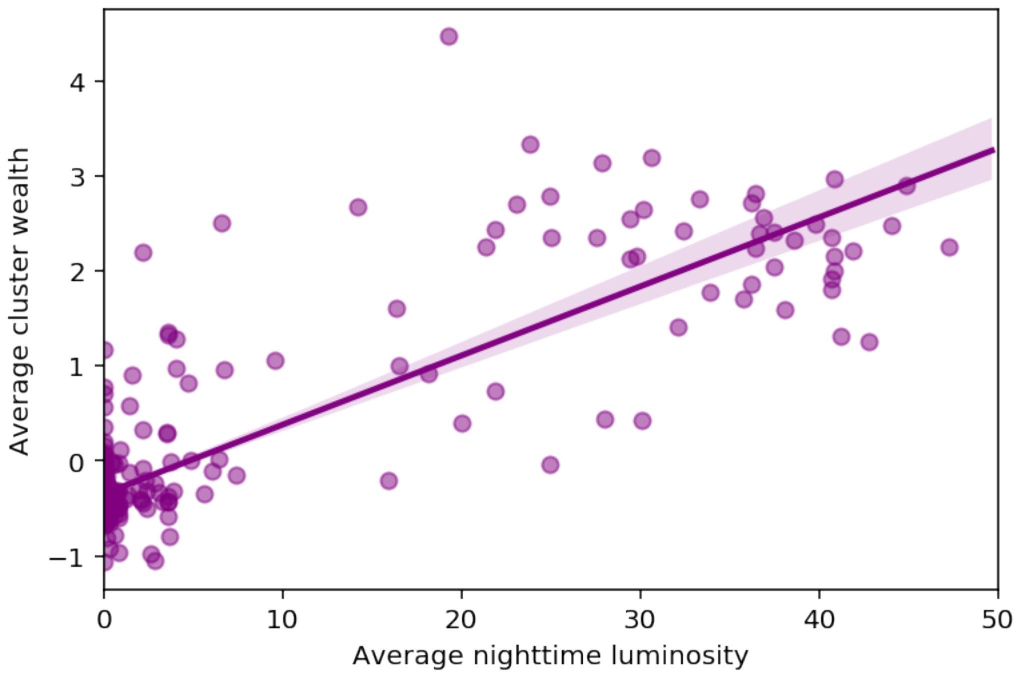
[8]:
```python
raster_file = 'data/nighttime_image/F182010.v4d_web.stable_lights.avg_vis.tif'
x_size, top_left_x_coords, top_left_y_coords, centroid_x_coords,␣
 ↪centroid_y_coords, bands_data = read_raster(raster_file)

# Save the result in compressed format - see https://docs.scipy.org/doc/numpy/
 ↪reference/generated/numpy.savez.html
```

```
np.savez('intermediate_files/nightlight.npz',␣
↪top_left_x_coords=top_left_x_coords, top_left_y_coords=top_left_y_coords,␣
↪bands_data=bands_data)
```

[10]:
```
# Get nightlight features for each cluster
def get_nightlight_feature(sample):
    idx, wealth, x, y = sample
    lon_idx, lat_idx = get_cell_idx(x, y, top_left_x_coords, top_left_y_coords)
    # Select the 10 * 10 pixels
    left_idx = lon_idx - 5
    right_idx = lon_idx + 4
    up_idx = lat_idx - 5
    low_idx = lat_idx + 4
    luminosity_100 = []
    for i in range(left_idx, right_idx + 1):
        for j in range(up_idx, low_idx + 1):
            # Get the luminosity of this pixel
            luminosity = bands_data[j, i, 0]
            luminosity_100.append(luminosity)
    luminosity_100 = np.asarray(luminosity_100)
    max_ = np.max(luminosity_100)
    min_ = np.min(luminosity_100)
    mean_ = np.mean(luminosity_100)
    median_ = np.median(luminosity_100)
    std_ = np.std(luminosity_100)
    return pd.Series({'id': idx, 'max_': max_, 'min_': min_, 'mean_': mean_,
                      'median_': median_, 'std_': std_, 'wealth': wealth})

clusters = pd.read_csv('intermediate_files/rwanda_cluster_avg_asset_2010.csv')
data_all = clusters.apply(lambda x: get_nightlight_feature([x['cluster'],␣
↪x['wlthindf'], x['longitude'], x['latitude']]), axis=1)
data_all.to_csv('intermediate_files/DHS_nightlights.csv', index=None)
```

[206]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'

data_all = pd.read_csv("DHS_nightlights.csv")

ax = sns.regplot(x="mean_", y="wealth", data=data_all, color='purple',␣
↪scatter_kws={'alpha':0.5})
plt.xlabel('Average nighttime luminosity')
plt.ylabel('Average cluster wealth')
plt.xlim([0, 50])
```

[206]: (0, 50)



[207]:
```
data_all.head()
```

[207]:

	id	max_	min_	mean_	median_	std_	wealth
0	1.0	6.0	0.0	0.06	0.0	0.596992	-0.531405
1	2.0	0.0	0.0	0.00	0.0	0.000000	-0.409830
2	3.0	0.0	0.0	0.00	0.0	0.000000	-0.478115
3	4.0	0.0	0.0	0.00	0.0	0.000000	-0.435960
4	5.0	0.0	0.0	0.00	0.0	0.000000	-0.449480

[210]:
```
from scipy import stats
print(stats.linregress(data_all['mean_'], data_all['wealth']))
```

```
LinregressResult(slope=0.07268134920271797, intercept=-0.3465173128978627,
rvalue=0.8607981773134475, pvalue=7.775683470696055e-146,
stderr=0.0019413136080220536)
```

4.2 3.2. Fit a model of wealth as a function of nightlights

- **INPUT**:
- `DHS_nightlights.csv`, from Step 3.1
- **OUTPUT**:
- R^2 of model

Above, we fit a regression line to illustrate the relationship between cluster average wealth and corresponding cluster nightlights. Now, we use cross-validation to get a better sense of out of sample accuracy.

[ ]:
```
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge

data_all = pd.read_csv('intermediate_files/DHS_nightlights.csv')
data_all = data_all[['max_', 'min_', 'mean_', 'median_', 'std_', 'wealth']].
 ↪values
np.random.seed(123)

alphas_list = np.logspace(-1, 5, 7)
final = []
for alpha in alphas_list:
    kf = KFold(n_splits=10, shuffle=True)
    scores = []
    for train_index, test_index in kf.split(data_all):
        reg = Ridge(alpha=alpha)
        train = data_all[train_index]
        test = data_all[test_index]
        reg.fit(train[:, :-1], train[:, -1])
        s = reg.score(test[:, :-1], test[:, -1])
        scores.append(s)
    final.append(np.mean(scores))

#print('R^2 of the best model: {:.3f}'.format(np.max(final)))
```

5 4. Download daytime satellite imagery

- **INPUT**:
- Google Maps API key
- `Sector_Boundary_2012.shp`: Rwandan shapefile
- **OUTPUT**:
- Thousands of satellite images (store in directory `google_image/`)

We will use the Google Static Maps API to download satellite images. Refer Google Static Maps introduction and Google Static Maps API Usage Limits. An API key is required before downloading.

We download the images from Google at zoom level 16 (pixel resolution is about 2.5m). We et the image size to be 400 pixels by 400 pixels, so that each image downloaded will cover 1 square kilometer. In this way, each daytime image downloaded will correspond to a single pixel from the nighttime imagery from Step 1 above.

[21]:
```
# Helper function to read a shapefile
def get_shp_extent(shp_file):
    """
    Function
```

```
    --------
    get_shp_extent

    Given a shapefile, get the extent (boundaries)

    Parameters
    ----------
    shp_file : string
        Path to the shapefile

    Returns
    -------
    extent : tuple
        Boundary location of the shapefile (x_min, x_max, y_min, y_max)
    """
    inDriver = ogr.GetDriverByName("ESRI Shapefile")
    inDataSource = inDriver.Open(inShapefile, 0)
    inLayer = inDataSource.GetLayer()
    extent = inLayer.GetExtent()
    # x_min_shp, x_max_shp, y_min_shp, y_max_shp = extent
    return extent
```

[23]:

```
# Helper functions to download images from Google Maps API

import io
import matplotlib
from matplotlib.pyplot import imread
import imageio
from retrying import retry

#@retry(wait_exponential_multiplier=1000, wait_exponential_max=3600000)
def save_img(url, file_path, file_name):
    """
    Function
    --------
    save_img

    Given a url of the map, save the image

    Parameters
    ----------
    url : string
        URL of the map from Google Map Static API
    file_path : string
        Folder name of the map
    file_name : string
        File name
```

```
    Returns
    -------
    None
    """
    a = urllib.request.urlopen(url).read()
    b = io.BytesIO(a)
    image = matplotlib.pyplot.imread(b)

    # When no image exists, the API will return an image with the same color.
    # In the center of such an image, 'Sorry. We have no imagery here' would be␣
↪displayed.
    # We should drop these images if large area of the image has the same color.
    if np.array_equal(image[:,:10,:],image[:,10:20,:]):
        pass
    else:
        imageio.imwrite(file_path + file_name, image[50:450, :, :])
```

[24]:
```
# Signing a URL using a URL signing secret

from staticmaps_signature import StaticMapURLSigner

def sign_url(api_key, secret, input_url):
    staticmap_url_signer = StaticMapURLSigner(public_key=api_key,␣
↪private_key=secret)
    signed_url = staticmap_url_signer.sign_url(input_url)
    return signed_url
```

[ ]:
```
import imageio.core.util
def silence_imageio_warning(*args, **kwargs):
    pass
imageio.core.util._precision_warn = silence_imageio_warning

# Now read in the shapefile for Rwanda and extract the edges of the country
inShapefile = "data/shp/Sector_Boundary_2012/Sector_Boundary_2012.shp"
x_min_shp, x_max_shp, y_min_shp, y_max_shp = get_shp_extent(inShapefile)

left_idx, top_idx = get_cell_idx(x_min_shp, y_max_shp, top_left_x_coords,␣
↪top_left_y_coords)
right_idx, bottom_idx = get_cell_idx(x_max_shp, y_min_shp, top_left_x_coords,␣
↪top_left_y_coords)

print((right_idx - left_idx)*(bottom_idx - top_idx)) # total number of images
print('left:', left_idx)
print('right:', right_idx)
print('top:', top_idx)
print('bottom:', bottom_idx)
```

```
YOUR_API_KEY = 'AIzaSyA5QSRCMBKr7AqugVybRPmfnwO6DQTsklU'
YOUR_SECRET = 'O3DROV4E4jhCi82R7JOrcwA4obI='

m = 1
for i in range(left_idx, right_idx + 1):
    for j in range(top_idx, bottom_idx + 1):
        lon = centroid_x_coords[i]
        lat = centroid_y_coords[j]

        url_to_sign = 'https://maps.googleapis.com/maps/api/staticmap?center='␣
↪+ str(lat) + ',' + \
               str(lon) + '&zoom=16&size=400x400&maptype=satellite'

        url = sign_url(YOUR_API_KEY, YOUR_SECRET, url_to_sign)

        lightness = bands_data[j, i, 0]
        file_path = 'google_image/' + str(lightness) + '/'
        if not os.path.isdir(file_path):
            os.makedirs(file_path)
        file_name = str(i) + '_' + str(j) +'.jpg'
        save_img(url, file_path, file_name)
        if m % 100 == 0:
            print(m)
        m += 1

print('Finished!')
```

6 5. Test whether basic features of daytime imagery can predict wealth

In step 3, we tested whether nightlight imagery could predict the wealth of Rwandan villages. We will now test whether daytime imagery can predict village wealth. We start by extracting simple metrics from the daytime imagery; in step 6 we will use more sophsticated methods to engineer these features from the images.

6.1 5.1. Extract "basic" features from daytime imagery

- **INPUT**:
- `google_image/...`: Raw images, from Step 4
- **OUTPUT**:
- `google_image_features_basic.csv`: Image features

We convert the raw data from the satellite imagery into a set of features that can be used in a machine learning algorithm.

[28]:
```
images_name = []
for i in range(64):
    dir_ = 'google_image/' + str(i) + '/'
    image_files = os.listdir(dir_)
    images_name.append(image_files)

def get_image_basic_feature(image_file):
    #image = ndimage.imread(image_file, mode='RGB')
    image = matplotlib.pyplot.imread(image_file)

    features = []
    for i in range(3):
        image_one_band = image[:, :, i].flatten()
        features.append(image_one_band)
    features = np.asarray(features)
    max_ = np.max(features, axis=1)
    min_ = np.min(features, axis=1)
    mean_ = np.mean(features, axis=1)
    median_ = np.median(features, axis=1)
    std_ = np.std(features, axis=1)
    return np.concatenate([max_, min_, mean_, median_, std_]).tolist()

feature_all = []
a = 0
t1 = time.time()
for i, images in enumerate(images_name):
    path = 'google_image/' + str(i) + '/'
    for image in images:
        x, y = [int(idx) for idx in image[:-4].split('_')]
        file_ = path + image
        feature = get_image_basic_feature(file_)
        feature = [x, y] + feature
        feature_all.append(feature)
        if a % 10000 == 0:
            t2 = time.time()
            print(a)
            print(t2 - t1)
            t1 = time.time()
        a += 1

feature_all = np.asarray(feature_all)
np.savetxt('intermediate_files/google_image_features_basic.csv', feature_all)
```

```
0
0.018002271652221680
10000
82.45701789855957
```

```
20000
88.55964493751526
30000
92.9567461013794
40000
88.00133085250854
50000
99.6015739440918
```

6.2 5.2. Merge daytime images with DHS data

- **INPUT**:
- `google_image_features_basic.csv`: Satellite imagery features, from Step 5.1
- `rwanda_cluster_avg_asset_2010.csv`: DHS cluster averages, from Step 2
- **OUTPUT**: Merged datasets
- `data/model/DHS_daytime.csv`: Merged dataset with 492 rows, and 16 columns (one indicates average cluster wealth, 15 daytime image features)

[29]:
```
features_basic = np.loadtxt('intermediate_files/google_image_features_basic.
↪csv')

def get_daytime_feature(sample):
    idx, wealth, x, y = sample
    lon_idx, lat_idx = get_cell_idx(x, y, top_left_x_coords, top_left_y_coords)
    left_idx = lon_idx - 5
    right_idx = lon_idx + 4
    up_idx = lat_idx - 5
    low_idx = lat_idx + 4
    features_100 = []
    for i in range(left_idx, right_idx + 1):
        for j in range(up_idx, low_idx + 1):
            feature = features_basic[((features_basic[:,0]==i) &␣
↪(features_basic[:,1]==j)),][:,2:]
            if len(feature) > 0:
                features_100.append(feature)
    if len(features_100) == 0:
        return np.asarray([np.nan] * 15 + [wealth]).tolist()
    else:
        features_all = np.concatenate(features_100, axis=0)
        mean_ = np.mean(features_all, axis=0).tolist()
        mean_.append(wealth)
        return mean_

clusters = pd.read_csv('intermediate_files/rwanda_cluster_avg_asset_2010.csv')
clusters['feature'] = clusters.apply(lambda x:␣
↪get_daytime_feature([x['cluster'], x['wlthindf'], x['longitude'],␣
↪x['latitude']]), axis=1)
```

```
data_all = clusters['feature']
data_all = np.asarray([i for i in data_all])
data_all = data_all[~np.isnan(data_all).any(axis=1)]

np.savetxt('intermediate_files/DHS_daytime.csv', data_all)
```

6.3 5.3. Fit a model of wealth as a function of basic daytime features

- **INPUT**:
- data/model/DHS_daytime.csv, from Step 5.2
- **OUTPUT**:
- R^2 of model

[30]:
```
data_all = np.loadtxt('DHS_daytime.csv')
alphas_list = np.logspace(-1, 5, 7)
final = []
for alpha in alphas_list:
    kf = KFold(n_splits=10, shuffle=True)
    scores = []
    for train_index, test_index in kf.split(data_all):
        reg = Ridge(alpha=alpha)
        train = data_all[train_index]
        test = data_all[test_index]
        reg.fit(train[:, :-1], train[:, -1])
        s = reg.score(test[:, :-1], test[:, -1])
        scores.append(s)
    final.append(np.mean(scores))

print('R^2 of the best model: {:.3f}'.format(np.max(final)))
```

```
R^2 of the best model: 0.533
```

[164]:
```
# Ridge Regression (basic daytime image features)
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge

predictions = [] # added
observations = [] # added
coefs = [] # added

data_all = np.loadtxt('DHS_daytime.csv')
alphas_list = np.logspace(-1, 5, 7)
final = []
for alpha in alphas_list:
    kf = KFold(n_splits=2, shuffle=True)
```

```
    scores = []
    for train_index, test_index in kf.split(data_all):
        reg = Ridge(alpha=alpha)
        train = data_all[train_index]
        test = data_all[test_index]
        reg.fit(train[:, :-1], train[:, -1])

        prediction = reg.predict(test[:, :-1]) # added
        predictions.append(prediction) # added
        observations.append(test[:, -1]) # added

        print(alpha)
        coefs.append(reg.coef_) # added

        s = reg.score(test[:, :-1], test[:, -1])
        scores.append(s)
    final.append(np.mean(scores))
    #print(coefs)

print('R^2 of the best model: {:.3f}'.format(np.max(final)))
```

```
0.1
0.1
1.0
1.0
10.0
10.0
100.0
100.0
1000.0
1000.0
10000.0
10000.0
100000.0
100000.0
R^2 of the best model: 0.558
```

[165]:
```
print(coefs)
```

```
[array([-0.56472942,  0.        , -0.78185828, -0.94981891,  0.44703017,
       -0.07567072,  0.02125057, -0.0423197 ,  0.14458589,  0.00361331,
       -0.01218081, -0.11937321,  0.09346289, -0.30396592,  0.34259884]),
array([-0.07320938,  0.        , -0.02591138, -0.38551225,  0.28806   ,
       -0.14649013,  0.14926343, -0.18154688,  0.17132659, -0.11150426,
        0.11022824, -0.13927787,  0.03853485, -0.26995293,  0.34766097]),
array([ 0.01874275,  0.        , -0.14422363, -0.54305808,  0.05222646,
        0.01395584,  0.13293688, -0.1395491 ,  0.16743118, -0.09487686,
        0.07821802, -0.14281085, -0.01542691, -0.17277105,  0.29943725]),
```

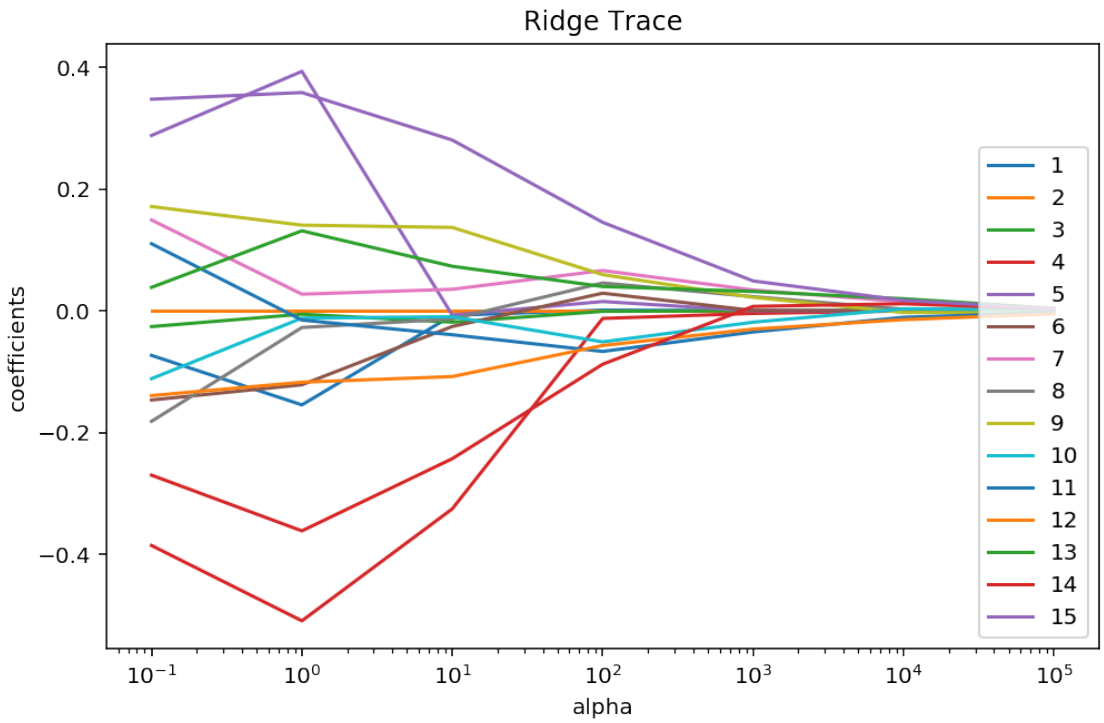
```
array([-0.15451883,  0.        , -0.00551213, -0.50931437,  0.39341569,
       -0.1214942 ,  0.02735779, -0.02739187,  0.1407582 , -0.01202651,
       -0.01517633, -0.11727338,  0.13146788, -0.36153071,  0.35859526]),
array([-0.00036912,  0.        ,  0.01891724, -0.07965818,  0.00258499,
       -0.00418671,  0.06456271,  0.0730683 ,  0.04867664, -0.05740306,
       -0.07797679, -0.04921761,  0.02435749, -0.20122663,  0.28387676]),
array([-0.00742849,  0.        , -0.01866387, -0.32530802, -0.00697526,
       -0.02570782,  0.03538064, -0.01336511,  0.13701946, -0.00972545,
       -0.0393124 , -0.10798059,  0.07316415, -0.24292379,  0.28056164]),
array([-0.00044183,  0.        , -0.00194037, -0.05700214, -0.02980909,
       -0.04460446,  0.03155145,  0.07106808,  0.07390955, -0.01384931,
       -0.09049506, -0.06653801,  0.02835003, -0.10027823,  0.16491082]),
array([ 0.00120716,  0.        , -0.00088585, -0.01248912,  0.01527537,
        0.02897133,  0.06619595,  0.04579588,  0.05943122, -0.05104208,
       -0.06671556, -0.05695564,  0.03991982, -0.0877242 ,  0.14536347]),
array([ 0.00011011,  0.        , -0.00081732, -0.00423671,  0.0018398 ,
        0.00444812,  0.03789356,  0.02056458,  0.02689136, -0.0128216 ,
       -0.04723195, -0.03103267,  0.03336763, -0.00146787,  0.05981218]),
array([ 9.70528746e-05,  0.00000000e+00,  2.28932125e-04, -4.26148935e-03,
        1.15797835e-04,  6.01662150e-04,  3.41930442e-02,  2.33251333e-02,
        2.21177330e-02, -1.87052716e-02, -3.47772639e-02, -3.00295803e-02,
        3.18095902e-02,  7.30941451e-03,  4.90696889e-02]), array([
1.93040778e-05,  0.00000000e+00, -5.14774651e-05, -3.71681006e-04,
       -2.83566856e-04, -6.38383206e-05,  1.54785520e-02,  1.87773989e-03,
       -2.84004789e-03,  4.12927217e-03, -1.03676431e-02, -1.42871652e-02,
        2.26627861e-02,  1.57374383e-02,  1.98096826e-02]), array([
2.45594309e-05,  0.00000000e+00, -4.49473409e-05, -2.83436937e-04,
        1.47422318e-04,  3.47584146e-04,  1.48807637e-02,  2.56167035e-03,
       -2.72538871e-03,  3.27046587e-03, -1.06421486e-02, -1.45262665e-02,
        1.98592313e-02,  1.16303103e-02,  1.72712064e-02]), array([
5.57078781e-06,  0.00000000e+00, -1.78535166e-05, -2.34156469e-05,
       -1.29784167e-04, -1.34363749e-04,  4.01107756e-03, -6.30683747e-04,
       -2.85923608e-03,  2.35314755e-03, -2.61356479e-03, -4.78892086e-03,
        4.56335529e-03,  3.15719347e-03,  3.52037782e-03]), array([
5.49779420e-06,  0.00000000e+00, -1.38329791e-05, -1.95850013e-05,
       -1.07393519e-04, -9.15102688e-05,  3.40580995e-03, -9.66868420e-04,
       -2.99175410e-03,  1.85366165e-03, -2.90707502e-03, -4.84780857e-03,
        4.17070339e-03,  2.89282136e-03,  3.35333371e-03])]
```

[200]:

```python
coefs_new_y = [coefs[i] for i in range(len(coefs)) if i % 2 != 0]
coefs_new_x = [coefs[i] for i in range(len(coefs)) if i % 2 == 0]

print(len(coefs_new_y[0]))
print(alphas_list)
```

```
15
[1.e-01 1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05]
```

[205]:
```
plt.figure(figsize=(8,5))
plt.plot(alphas_list, coefs_new_y)
plt.xscale('log')
#plt.xlim(plt.get_xlim()[::-1])  # reverse axis
#plt.set_xlim(10e-10, 10e5)
plt.xlabel('alpha')
plt.ylabel('coefficients')
plt.title('Ridge Trace') #Ridge coefficients as a function of the regularization
plt.legend(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',␣
↪'13', '14', '15'], loc='lower right')
plt.axis('tight')
plt.show()
```



[145]:
```
print(len(coefs))
print(coefs[0])
print(len(alphas_list))
print(alphas_list)
print(final)
```

```
14
[-1.35798261  0.          0.51337732 -0.73429653  0.41293064 -0.20213816
  0.01668903 -0.17706966  0.3693795   0.01649106  0.10674706 -0.32987147
  0.04453044 -0.24822232  0.31583701]
7
```

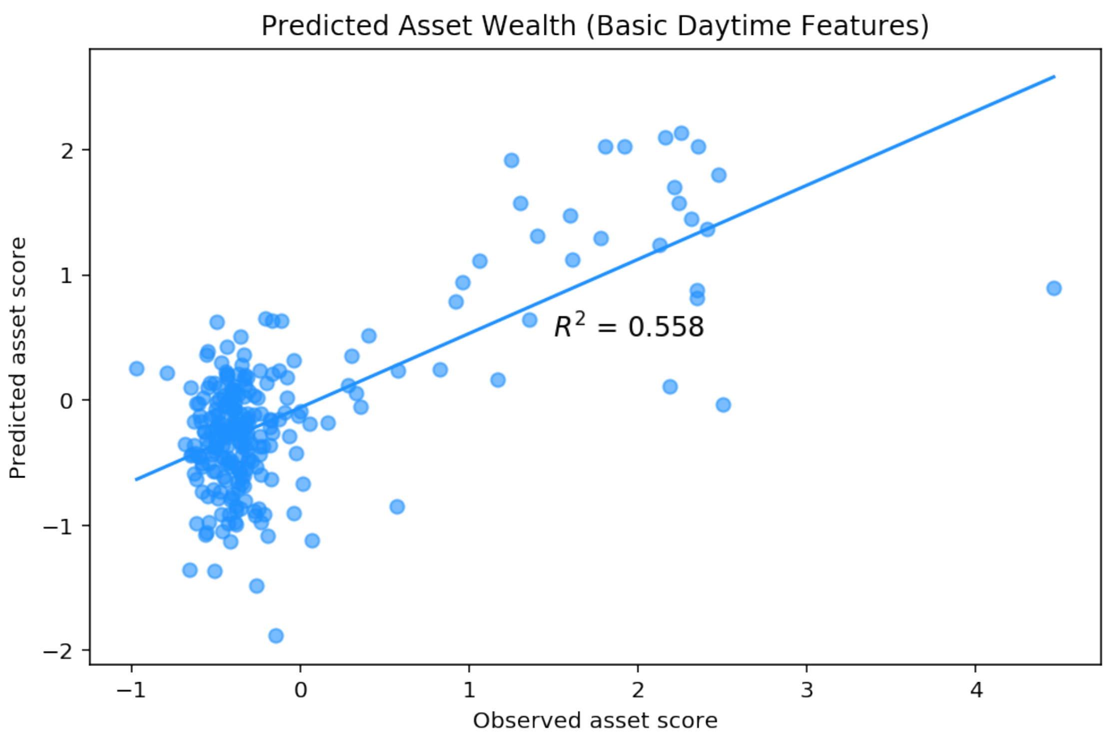
```
[1.e-01 1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05]
[0.5332090891543441, 0.5266915532321286, 0.5573468657104865, 0.5375323034411684,
0.4596512368012614, 0.34439650120065346, 0.12889187575310806]
```

[149]:
```
star_prediction = predictions[2]
star_observation = observations[2]

print(len(star_prediction))
print(len(data_all))
```

```
246
492
```

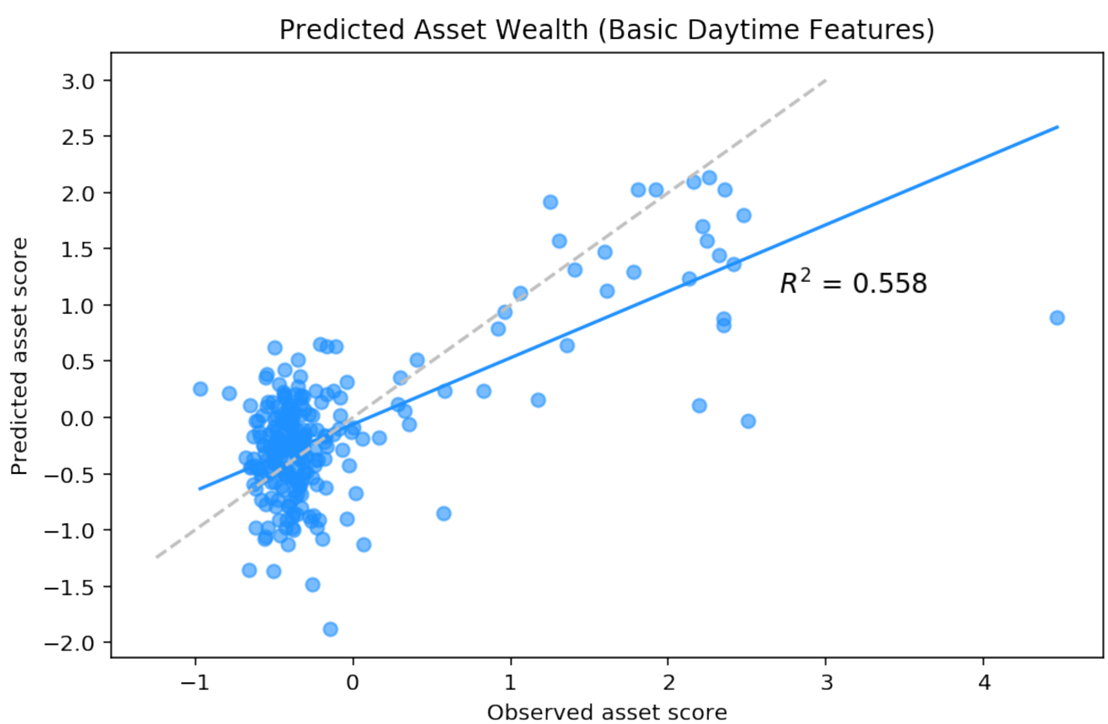
[132]:
```
plt.figure(figsize=(8,5))
plt.scatter(star_observation, star_prediction, alpha=0.6, color='dodgerblue')
plt.plot(np.unique(star_observation), np.poly1d(np.polyfit(star_observation,
→star_prediction, 1))(np.unique(star_observation)), color='dodgerblue')
plt.text(1.5, 0.5, '$R^2$ = 0.558', size=12)
plt.xlabel('Observed asset score')
plt.ylabel('Predicted asset score')
plt.title('Predicted Asset Wealth (Basic Daytime Features)')
```

[132]: Text(0.5, 1.0, 'Predicted Asset Wealth (Basic Daytime Features)')



[138]:
```
plt.figure(figsize=(8,5))
plt.scatter(star_observation, star_prediction, alpha=0.6, color='dodgerblue')
plt.plot(np.unique(star_observation), np.poly1d(np.polyfit(star_observation,␣
 ↪star_prediction, 1))(np.unique(star_observation)), color='dodgerblue')
plt.text(2.7, 1.1, '$R^2$ = 0.558', size=12)
plt.xlabel('Observed asset score')
plt.ylabel('Predicted asset score')
plt.yticks(np.arange(-2.5, 3.01, 0.5))
plt.title('Predicted Asset Wealth (Basic Daytime Features)')

plt.plot([max(plt.xlim()[0], plt.ylim()[0]),
          min(plt.xlim()[1], plt.ylim()[1])],
         [max(plt.xlim()[0], plt.ylim()[0]),
          min(plt.xlim()[1], plt.ylim()[1])], color='silver', linestyle="--")
```

[138]:
```
[<matplotlib.lines.Line2D at 0x7f08e26e0310>]
```



7 6. Extract features from daytime imagery using deep learning libraries

7.1 6.1. Use the keras library to use a basic CNN to extract features of the daytime images

- **INPUT**:
- `google_image/...`: Raw images, from Step 4
- **OUTPUT**:
- `google_image_features_cnn.csv`: Image features

We begin by using a Convolutional Neural Network (CNN) that has been pre-trained on ImageNet to extract features from the images. We used the `Keras` library, which provides a very straightforward interface to TensorFlow.

[ ]:
```
from keras.applications.vgg16 import VGG16
import numpy as np
from keras.preprocessing import image
from keras.models import Sequential
from tensorflow.keras.applications.vgg16 import decode_predictions,␣
↪preprocess_input

from keras.layers.convolutional import Convolution2D, AveragePooling2D
from keras.optimizers import SGD
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Dropout
from multiprocessing import Pool
import os
import time
import pandas as pd
import numpy as np
from keras.models import Model

images_name = {}
for i in range(64):
    dir_ = 'google_image/' + str(i) + '/'
    image_files = os.listdir(dir_)
    for f in image_files:
        images_name[f] = i

def get_cell_idx(lon, lat, top_left_x_coords, top_left_y_coords):
    lon_idx = np.where(top_left_x_coords < lon)[0][-1]
    lat_idx = np.where(top_left_y_coords > lat)[0][-1]
    return lon_idx, lat_idx
```

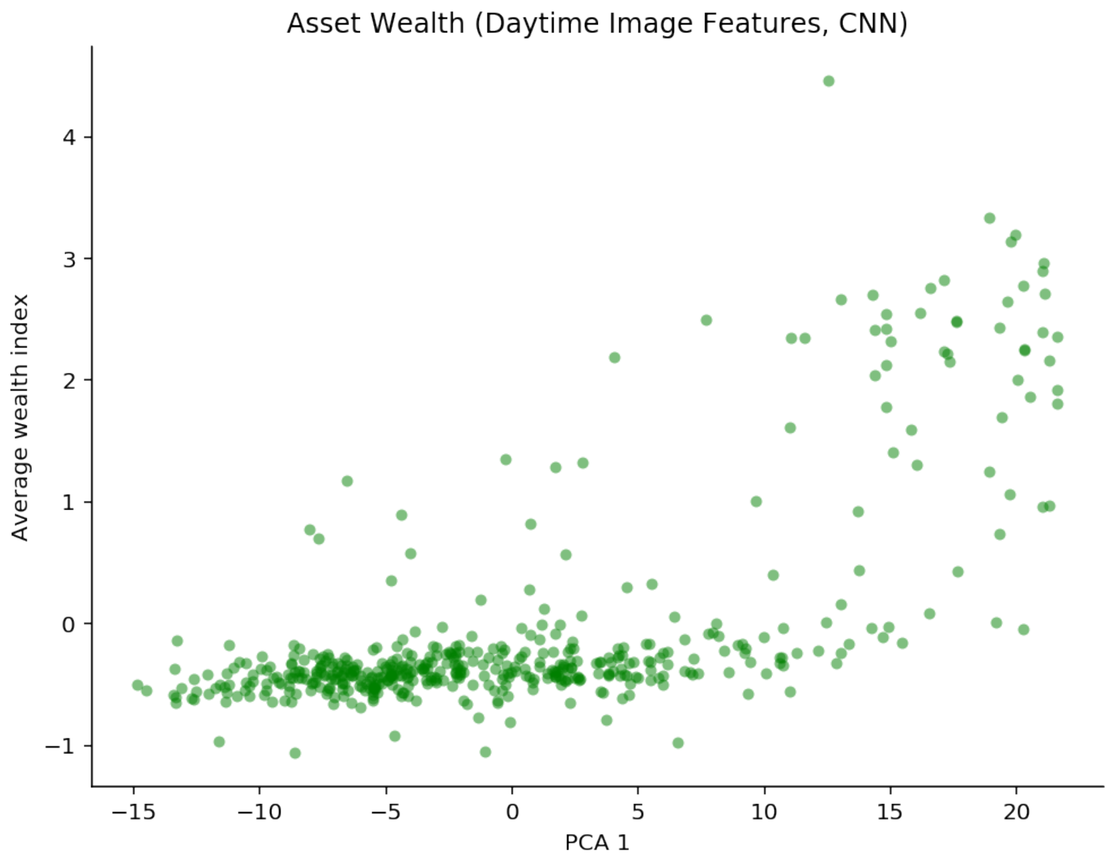
```python
npzfile = np.load('intermediate_files/nightlight.npz')
print(npzfile.files)
top_left_x_coords = npzfile['top_left_x_coords']
top_left_y_coords = npzfile['top_left_y_coords']
bands_data = npzfile['bands_data']

# Get image featuers
base_model = VGG16(weights='imagenet')
model = Model(input=base_model.input, output=base_model.get_layer('fc2').output)

def get_input_feature(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    # img = image.load_img(img_path)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    features = model.predict(x)
    return features[0]

def get_daytime_feature(sample):
    idx, wealth, x, y = sample
    print(idx)
    lon_idx, lat_idx = get_cell_idx(x, y, top_left_x_coords, top_left_y_coords)
    left_idx = lon_idx - 5
    right_idx = lon_idx + 4
    up_idx = lat_idx - 5
    low_idx = lat_idx + 4
    features_100 = []
    for i in range(left_idx, right_idx + 1):
        for j in range(up_idx, low_idx + 1):
            file_name = str(i) + '_' + str(j) + '.jpg'
            if file_name in images_name:
                luminosity = images_name[file_name]
                feature = get_input_feature('google_image/' + str(luminosity) +␣
 ↪'/' + file_name)
                features_100.append(feature)
    if len(features_100) == 0:
        print('nononono: ' + str(idx))
        return np.asarray([np.nan] * 4096 + [wealth]).tolist()
    else:
        features_100 = np.asarray(features_100)
        return np.append(np.mean(features_100, axis=0), wealth).tolist()

clusters = pd.read_csv('intermediate_files/rwanda_cluster_avg_asset_2010.csv')
clusters['feature'] = clusters.apply(lambda x:␣
 ↪get_daytime_feature([x['cluster'], x['wlthindf'], x['longitude'],␣
 ↪x['latitude']]), axis=1)
```

```
data_all = clusters['feature']
data_all = np.asarray([i for i in data_all])
data_all = data_all[~np.isnan(data_all).any(axis=1)]

np.savetxt('intermediate_files/google_image_features_cnn.csv', data_all)
```

8 6.2. Test whether these new features of satellite imagery can predict wealth

- **INPUT**:
- `google_image_features_cnn.csv`: Satellite imagery features, from Step 6.1
- `rwanda_cluster_avg_asset_2010.csv`: DHS cluster averages, from Step 2
- **OUTPUT**:
- `data/model/DHS_daytime.csv`: Merged dataset with 492 rows, and 4097 columns (one indicates average cluster wealth, 4096 CNN-based features)
- R^2 of model

[24]:
```
from sklearn.decomposition import PCA
import numpy as np

data_all = np.loadtxt('google_image_features_cnn.csv')
x = data_all[:, :-1]
y = data_all[:, -1]
pca = PCA(n_components=1)
pca.fit(x)
proj = pca.transform(x)

fig, ax = plt.subplots(figsize=(8,6))
ax.plot(proj[:,0], y, 'o', c='green', markersize=5, markeredgecolor='none',␣
 ↪alpha=0.5)
plt.xlabel('PCA 1')
plt.ylabel('Average wealth index')
plt.title('Asset Wealth (Daytime Image Features, CNN)')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')
```



9 7. Transfer Learning

We propose a "transfer learning" step. In other words, instead of using the image features extracted by the CNN directly, we want to retrain the CNN to predict nightlights from daytime imagery, and use those features, which presumably are more appropriate to our final prediction task.

9.1 7.1. Use the nightlights to retrain the CNN and extract features

- **INPUT**:
- `google_image/...`: Raw images, from Step 4
- **OUTPUT**:
- `google_image_features_cnn_retrained.csv`: Image features

We first divide daytime images into three groups, corresponding to images where the corresponding night-lights pixel is dim, medium, or bright. We use these values to define groups: [0, 3), [3, 35), [35, 64).

[44]:
```python
import shutil

def move_to_group(lightness_small, lightness_big, class_id):
    new_directory = 'google_image_cnn/class_' + str(class_id) + '/'
    if not os.path.isdir(new_directory):
        os.makedirs(new_directory)
    for i in range(lightness_small, lightness_big):
        path = 'google_image/' + str(i) + '/'
        for f in os.listdir(path):
            shutil.copyfile(path + f, new_directory + f)

move_to_group(0, 3, 1)
move_to_group(3, 35, 2)
move_to_group(35, 64, 3)
```

[51]:
```python
# Checking number of images in each group

dim_list = os.listdir('google_image_cnn/class_1'); print('dim:', len(dim_list))
medium_list = os.listdir('google_image_cnn/class_2'); print('medium:',␣
↪len(medium_list))
bright_list = os.listdir('google_image_cnn/class_3'); print('bright:',␣
↪len(bright_list))
```

```
dim: 51250
medium: 1740
bright: 146
```

9.2 7.2. Test whether "deep" features of satellite imagery can predict wealth

- **INPUT**:
- `google_image_cnn/...`: Satellite images from 7.1
- **OUTPUT**:
- `data/model/DHS_CNN.csv`: Merged dataset with 492 rows, and 4097 columns (one indicates average cluster wealth, 4096 CNN features)
- R^2 of model

[53]:
```python
import split_folders

# Dividing up data into train and test sets, with an 80%-20% split
split_folders.ratio('google_image_cnn', output='google_image_cnn_train_test',␣
↪ratio=(.8, .2))
```

[56]:
```python
# Rename 'val' to 'test', and delte empty 'test' folder created in previous step
os.rmdir('google_image_cnn_train_test/test')
os.rename('google_image_cnn_train_test/val', 'google_image_cnn_train_test/test')

# Train-test split results
```

```
dim_list_train = os.listdir('google_image_cnn_train_test/train/class_1');␣
↪print('dim_train:', len(dim_list_train))
dim_list_test = os.listdir('google_image_cnn_train_test/test/class_1');␣
↪print('dim_test:', len(dim_list_test))
medium_list_train = os.listdir('google_image_cnn_train_test/train/class_2');␣
↪print('medium_train:', len(medium_list_train))
medium_list_test = os.listdir('google_image_cnn_train_test/test/class_2');␣
↪print('medium_test:', len(medium_list_test))
bright_list_train = os.listdir('google_image_cnn_train_test/train/class_3');␣
↪print('bright_train:', len(bright_list_train))
bright_list_test = os.listdir('google_image_cnn_train_test/test/class_3');␣
↪print('bright_test:', len(bright_list_test))
```

```
dim_train: 41000
dim_test: 10250
medium_train: 1392
medium_test: 348
bright_train: 116
bright_test: 30
```

[ ]:
```
from keras.applications.vgg16 import VGG16
import numpy as np
from keras.preprocessing import image
from keras.models import Sequential
from tensorflow.keras.applications.vgg16 import decode_predictions,␣
↪preprocess_input

from keras.layers.convolutional import Convolution2D, AveragePooling2D
from keras.optimizers import SGD
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Dropout
from keras.models import Model
from multiprocessing import Pool
import os
import time

# Get image featuers
model_old = VGG16(weights='imagenet', include_top=False)

def get_input_feature(img_path):
    # img = image.load_img(img_path, target_size=(400, 400))
    img = image.load_img(img_path)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
```

```
    features = model_old.predict(x)
    return features[0]

# Train
all_figures = []
trainLabels = []

# Need upsampling because of the unbalance of the training classes
path_1 = 'google_image_cnn_train_test/train/class_1/'
# path_1 = 'google_image_cnn/class_1/'
class_1_files = os.listdir(path_1)
trainLabels += [[1, 0, 0]] * len(class_1_files)
all_figures += [path_1 + i for i in class_1_files]

path_2 = 'google_image_cnn_train_test/train/class_2/'
# path_2 = 'google_image_cnn/class_2/'
class_2_files = os.listdir(path_2)
trainLabels += [[0, 1, 0]] * len(class_2_files)
all_figures += [path_2 + i for i in class_2_files]

path_3 = 'google_image_cnn_train_test/train/class_3/'
# path_3 = 'google_image_cnn/class_3/'
class_3_files = os.listdir(path_3)
trainLabels += [[0, 0, 1]] * len(class_3_files)
all_figures += [path_3 + i for i in class_3_files]

# a = get_input_feature(all_figures[0])
# pool = Pool(10)
# trainData = pool.map(get_input_feature, all_figures)

trainData = []
t1 = time.time()
for idx, i in enumerate(all_figures):
    a = get_input_feature(i)
    trainData.append(a)
    if idx % 1000 == 0:
        t2 = time.time()
        print(idx)
        print(t2 - t1)
        t1 = time.time()

x_all = np.asarray(trainData)
y_all = np.asarray(trainLabels)

# Test
all_figures = []
testLabels = []
```

```python
path_1 = 'google_image_cnn_train_test/test/class_1/'
# path_1 = 'google_image_cnn/class_1/'
class_1_files = os.listdir(path_1)
testLabels += [[1, 0, 0]] * len(class_1_files)
all_figures += [path_1 + i for i in class_1_files]

path_2 = 'google_image_cnn_train_test/test/class_2/'
# path_2 = 'google_image_cnn/class_2/'
class_2_files = os.listdir(path_2)
testLabels += [[0, 1, 0]] * len(class_2_files)
all_figures += [path_2 + i for i in class_2_files]

path_3 = 'google_image_cnn_train_test/test/class_3/'
# path_3 = 'google_image_cnn/class_3/'
class_3_files = os.listdir(path_3)
testLabels += [[0, 0, 1]] * len(class_3_files)
all_figures += [path_3 + i for i in class_3_files]

# a = get_input_feature(all_figures[0])
# pool = Pool(10)
# testData = pool.map(get_input_feature, all_figures)

testData = []
t1 = time.time()
for idx, i in enumerate(all_figures):
    a = get_input_feature(i)
    testData.append(a)
    if idx % 1000 == 0:
        t2 = time.time()
        print(idx)
        print(t2 - t1)
        t1 = time.time()

x_all_test = np.asarray(testData)
y_all_test = np.asarray(testLabels)

# np.savez('google_image_feature.npz', x_all=x_all, y_all=y_all)
np.savez('google_image_feature_train_test.npz', x_all=x_all, y_all=y_all,
         x_all_test=x_all_test, y_all_test=y_all_test)

# npzfile = np.load('google_image_feature_upsampling.npz')
# print npzfile.files
# x_all = npzfile['x_all']
# y_all = npzfile['y_all']

x_train = x_all
```

```
x_test = x_all_test
y_train = y_all
y_test = y_all_test

# The model configuration:
# https://github.com/nealjean/predicting-poverty/blob/master/model/
 ↪predicting_poverty_deploy.prototxt
model = Sequential()
model.add(Convolution2D(4096, 6, 6, activation='relu', input_shape=(12, 12,␣
 ↪512), subsample=(6, 6), name='input'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, 1, 1, activation='relu', subsample=(1, 1),␣
 ↪name='conv_7'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, 1, 1, subsample=(1, 1), name='conv_8'))
model.add(AveragePooling2D((2, 2), strides=(1, 1), name='add_pool'))

model.add(Flatten(name='flatten'))
model.add(Dense(3))
model.add(Activation("softmax"))

opt = SGD(lr=1e-2)
# model.compile(loss="categorical_crossentropy", optimizer='adam',␣
 ↪metrics=["accuracy"])
model.compile(loss="categorical_crossentropy", optimizer=opt,␣
 ↪metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=100, nb_epoch=10, verbose=1)

score = model.evaluate(x_test, y_test, verbose=0)
print(score)
```

[63]:
```
# Loading from saved representation

npzfile = np.load('google_image_feature_train_test.npz')
print(npzfile.files)
x_all = npzfile['x_all']
y_all = npzfile['y_all']
x_all_test = npzfile['x_all_test']
y_all_test = npzfile['y_all_test']

x_train = x_all
x_test = x_all_test
y_train = y_all
y_test = y_all_test
```

```
['x_all', 'y_all', 'x_all_test', 'y_all_test']
```

[79]:
```
print('Number of training images:', len(dim_list_train) +␣
↪len(medium_list_train) + len(bright_list_train))
print('Number of testing images:', len(dim_list_test) + len(medium_list_test) +␣
↪len(bright_list_test))

print('x_train:', x_train.shape)
print('x_test:', x_test.shape)
print('y_train:', y_train.shape)
print('y_test:', y_test.shape)
```

```
Number of training images: 42508
Number of testing images: 10628
x_train: (42508, 10, 12, 512)
x_test: (10628, 10, 12, 512)
y_train: (42508, 3)
y_test: (10628, 3)
```

[ ]:
```
# Get features
npzfile = np.load('intermediate_files/nightlight.npz')
print(npzfile.files)
top_left_x_coords = npzfile['top_left_x_coords']
top_left_y_coords = npzfile['top_left_y_coords']
bands_data = npzfile['bands_data']

def get_cell_idx(lon, lat, top_left_x_coords, top_left_y_coords):
    lon_idx = np.where(top_left_x_coords < lon)[0][-1]
    lat_idx = np.where(top_left_y_coords > lat)[0][-1]
    return lon_idx, lat_idx

model_select = Model(input=model.input, output=model.get_layer('add_pool').
↪output)

images_name = {}
for i in range(64):
    dir_ = 'google_image/' + str(i) + '/'
    image_files = os.listdir(dir_)
    for f in image_files:
        images_name[f] = i

def get_input_feature_2(img_path):
    # img = image.load_img(img_path, target_size=(400, 400))
    img = image.load_img(img_path)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    features = model_old.predict(x)
    pool_features = model_select.predict(features)
```

```
    return pool_features[0]

def get_daytime_feature(sample):
    idx, wealth, x, y = sample
    print(idx)
    lon_idx, lat_idx = get_cell_idx(x, y, top_left_x_coords, top_left_y_coords)
    left_idx = lon_idx - 5
    right_idx = lon_idx + 4
    up_idx = lat_idx - 5
    low_idx = lat_idx + 4
    features_100 = []
    for i in range(left_idx, right_idx + 1):
        for j in range(up_idx, low_idx + 1):
            file_name = str(i) + '_' + str(j) + '.jpg'
            if file_name in images_name:
                luminosity = images_name[file_name]
                feature = get_input_feature_2('google_image/' + str(luminosity)␣
 ↪+ '/' + file_name)
                features_100.append(feature)
    if len(features_100) == 0:
        print('nononono: ' + str(idx))
        return np.asarray([np.nan] * 4096 + [wealth]).tolist()
    else:
        features_100 = np.asarray(features_100)
        return np.append(np.mean(features_100, axis=0), wealth).tolist()

clusters = pd.read_csv('intermediate_files/rwanda_cluster_avg_asset_2010.csv')
clusters['feature'] = clusters.apply(lambda x:␣
 ↪get_daytime_feature([x['cluster'], x['wlthindf'],
                                                                      ␣
 ↪x['longitude'], x['latitude']]), axis=1)
data_all = clusters['feature']
data_all = np.asarray([i for i in data_all])
data_all = data_all[~np.isnan(data_all).any(axis=1)]

np.savetxt('intermediate_files/google_image_features_cnn_retrain.csv', data_all)
```

10 8. Construct a high-resolution map of the distribution of predicted wealth

- **INPUT**:
- Model, image features (data/model/features_all_predictimage_location.csv)
- **OUTPUT**:
- Heatmap characterizing poverty in Rwanda

We create a heatmap showing the distribution of predicted wealth in Rwanda.

[ ]:
```python
def array_to_raster(x_size, y_size, left, top, bands_data, no_data_value,␣
 ↪filename):
    driver = gdal.GetDriverByName('GTiff')

    dataset = driver.Create(
        filename,
        bands_data.shape[1],
        bands_data.shape[0],
        1,
        gdal.GDT_Float32, )

    dataset.SetGeoTransform((left, x_size, 0, top,0, y_size))

    # dataset.SetProjection(wgs84)
    outband = dataset.GetRasterBand(1)
    outband.WriteArray(bands_data[:, :, 0])
    outband.FlushCache()  # Write to disk.
    outband.SetNoDataValue(no_data_value)

features_all_predict = np.genfromtxt('data/model/features_all_predict.csv',␣
 ↪delimiter=',')
image_loc = np.genfromtxt('data/model/image_loc.csv', delimiter=',')

x_image_num = right_idx + 1 - left_idx
y_image_num = bottom_idx + 1 - top_idx

output_array = np.ndarray(shape=(y_image_num, x_image_num, 1))
no_data_value = 100
output_array.fill(no_data_value)

for idx, v in enumerate(image_loc):
    x, y = v
    x_id = x - left_idx
    y_id = y - top_idx
    output_array[y_id, x_id, 0] = features_all_predict[idx]

array_to_raster(x_size, - x_size, top_left_x_coords[left_idx],
                top_left_y_coords[top_idx], output_array, no_data_value,␣
 ↪'poverty_mapping.tiff')
```

Transfer Learning

January 10, 2020

[33]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.decomposition import PCA

import tensorflow
import keras
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, AveragePooling2D
from keras.optimizers import SGD
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Dropout
from keras.models import Model
from multiprocessing import Pool
from tensorflow.keras.applications.vgg16 import decode_predictions,␣
 ↪preprocess_input

from keras.callbacks import ModelCheckpoint
from keras import backend
backend.set_image_data_format('channels_first')
```

0.0.1 Model configuration (transfer learning)

[4]:
```
loaded = np.load('compressed_train_test.npz')
x_train = loaded['x_train']
x_test = loaded['x_test']
y_train = loaded['y_train']
y_test = loaded['y_test']
```

```
print("Decompressing complete.")
```

```
Decompressing complete.
```

Definition

[6]:
```
model = Sequential()
model.add(Convolution2D(4096, (6, 6), activation='relu', input_shape=(10, 12,␣
↪512), strides=(6, 6), name='input'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), activation='relu', strides=(1, 1),␣
↪name='conv_7'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), strides=(1, 1), name='conv_8'))
model.add(AveragePooling2D((2, 2), strides=(1, 1), name='add_pool'))

model.add(Flatten(name='flatten'))
model.add(Dense(3))
model.add(Activation("softmax"))
```

Compilation and Fitting

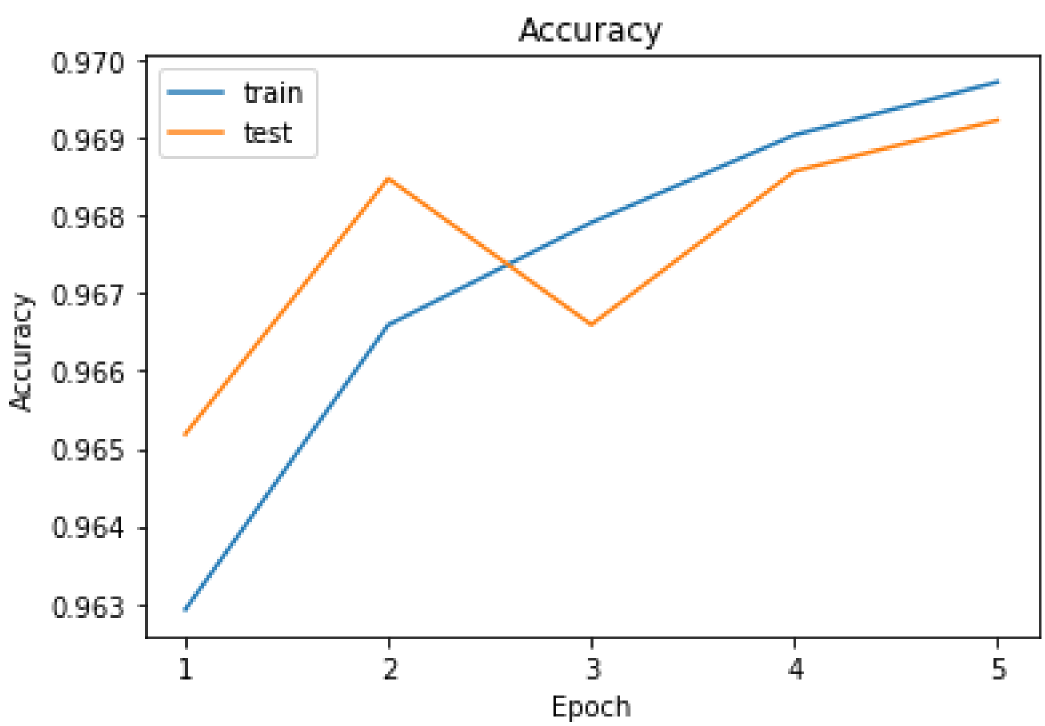
[10]:
```
opt = SGD(lr=1e-2)
#opt = keras.optimizers.Adam(lr=1.46e-3)
model.compile(loss="categorical_crossentropy", optimizer=opt,␣
↪metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=128, epochs=5,␣
↪validation_data=(x_test, y_test), shuffle=True, verbose=1)
```

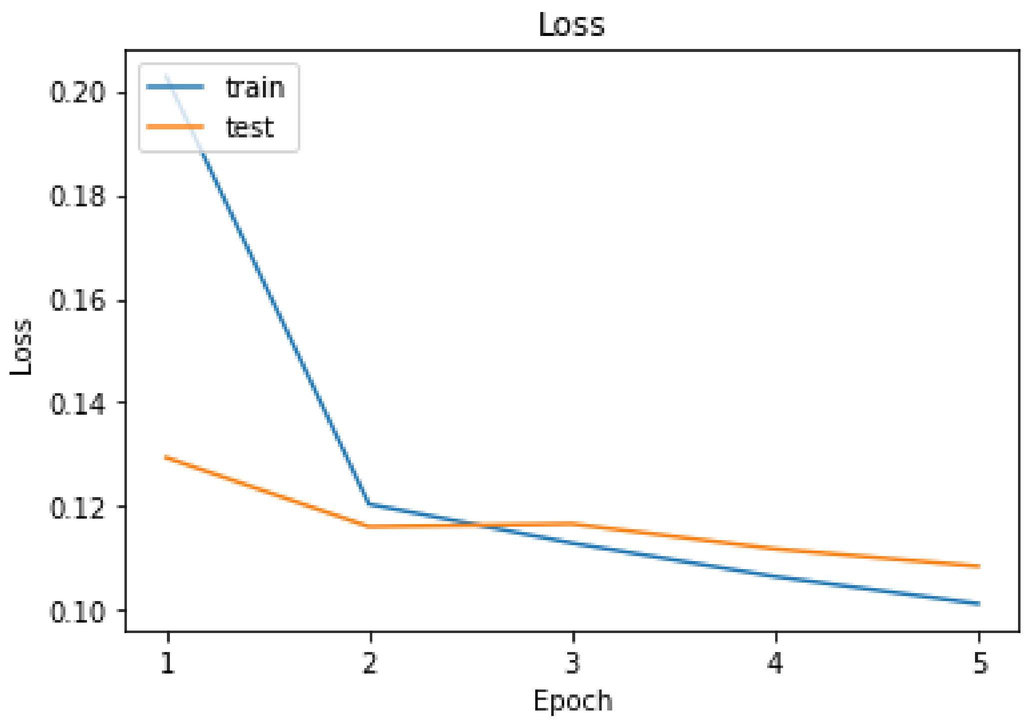
```
WARNING:tensorflow:From /opt/miniconda/envs/sugar/lib/python3.7/site-
packages/tensorflow/python/ops/math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 42508 samples, validate on 10628 samples
Epoch 1/5
42508/42508 [==============================] - 255s 6ms/step - loss: 0.2027 -
acc: 0.9629 - val_loss: 0.1294 - val_acc: 0.9652
Epoch 2/5
42508/42508 [==============================] - 336s 8ms/step - loss: 0.1204 -
acc: 0.9666 - val_loss: 0.1161 - val_acc: 0.9685
Epoch 3/5
42508/42508 [==============================] - 251s 6ms/step - loss: 0.1130 -
acc: 0.9679 - val_loss: 0.1167 - val_acc: 0.9666
Epoch 4/5
42508/42508 [==============================] - 234s 6ms/step - loss: 0.1065 -
```

```
acc: 0.9690 - val_loss: 0.1118 - val_acc: 0.9686
Epoch 5/5
42508/42508 [==============================] - 234s 6ms/step - loss: 0.1013 -
acc: 0.9697 - val_loss: 0.1085 - val_acc: 0.9692
```

Plotting accuracies and losses

[40]:
```
# Summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.xticks(np.arange(5), ('1', '2', '3', '4', '5'))
plt.show()

# Summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.xticks(np.arange(5), ('1', '2', '3', '4', '5'))
plt.show()
```





Evaluating

```
[14]: scores = model.evaluate(x_test, y_test, verbose=0)
      print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
acc: 96.92%
```

Saving model and architecture to single file

```
[15]: model.save("transfer_learning_model.h5")
      print("Saved model to disk.")
```

```
Saved model to disk.
```

0.0.2 Final model wealth prediction accuracies

After training the VGG on the images (using transfer learning) to predict the nightlight bins, we compute the 4096-size feature vector (right before it is condensed into classification) for each image and average these across a cluster.

Getting daytime image features

[ ]:
```
backend.set_image_data_format('channels_last')
model_old = VGG16(weights='imagenet', include_top=False)

npzfile = np.load('nightlight.npz')
print(npzfile.files)
top_left_x_coords = npzfile['top_left_x_coords']
top_left_y_coords = npzfile['top_left_y_coords']
bands_data = npzfile['bands_data']

def get_cell_idx(lon, lat, top_left_x_coords, top_left_y_coords):
    lon_idx = np.where(top_left_x_coords < lon)[0][-1]
    lat_idx = np.where(top_left_y_coords > lat)[0][-1]
    return lon_idx, lat_idx

model_select = Model(input=model.input, output=model.get_layer('add_pool').
 ↪output)

images_name = {}
for i in range(64):
    dir_ = 'google_image/' + str(i) + '/'
    image_files = os.listdir(dir_)
    for f in image_files:
        images_name[f] = i

def get_input_feature_2(img_path):
    # img = image.load_img(img_path, target_size=(400, 400))
    img = image.load_img(img_path)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    features = model_old.predict(x)
    pool_features = model_select.predict(features)
    return pool_features[0]

def get_daytime_feature(sample):
    idx, wealth, x, y = sample
    print(idx)
    lon_idx, lat_idx = get_cell_idx(x, y, top_left_x_coords, top_left_y_coords)
    left_idx = lon_idx - 5
    right_idx = lon_idx + 4
    up_idx = lat_idx - 5
    low_idx = lat_idx + 4
    features_100 = []
    for i in range(left_idx, right_idx + 1):
        for j in range(up_idx, low_idx + 1):
            file_name = str(i) + '_' + str(j) + '.jpg'
```

```
            if file_name in images_name:
                luminosity = images_name[file_name]
                feature = get_input_feature_2('google_image/' + str(luminosity)␣
↪+ '/' + file_name)
                features_100.append(feature)
    if len(features_100) == 0:
        print('nononono: ' + str(idx))
        return np.asarray([np.nan] * 4096 + [wealth]).tolist()
    else:
        features_100 = np.asarray(features_100)
        return np.append(np.mean(features_100, axis=0), wealth).tolist()

clusters = pd.read_csv('rwanda_cluster_avg_asset_2010.csv')
clusters['feature'] = clusters.apply(lambda x:␣
↪get_daytime_feature([x['cluster'], x['wlthindf'],
                                                                    ␣
↪x['longitude'], x['latitude']]), axis=1)
data_all = clusters['feature']
data_all = np.asarray([i for i in data_all])
data_all = data_all[~np.isnan(data_all).any(axis=1)]

np.savetxt('google_image_features_cnn_retrain.csv', data_all)
```

R^2 value of model

[29]:
```
data_all = np.loadtxt('google_image_features_cnn_retrain.csv')
print("Features loaded.")
```

```
Features loaded.
```

[34]:
```
alphas_list = np.logspace(-1, 5, 7)

final = []
for alpha in alphas_list:
    kf = KFold(n_splits=10, shuffle=True)
    scores = []
    for train_index, test_index in kf.split(data_all):
        reg = Ridge(alpha=alpha)
        train = data_all[train_index]
        test = data_all[test_index]
        train_x = train[:, :-1]
        train_y = train[:, -1]
        test_x = test[:, :-1]
        test_y = test[:, -1]
        # Reduce dimensions to avoid overfitting
        pca = PCA(n_components=100)
        pca.fit(train_x)
```

```
        train_x = pca.transform(train_x)
        test_x = pca.transform(test_x)
        reg.fit(train_x, train_y)
        s = reg.score(test_x, test_y)
        scores.append(s)
    final.append(np.mean(scores))

print('R^2 of the best model: {:.3f}'.format(np.max(final)))
```

```
R^2 of the best model: 0.718
```

[35]:
```
print(final)
```

```
[0.6571728522240462, 0.709800758403902, 0.6875553271555113, 0.711863772741731,
0.7183663582635443, 0.5483857781413544, 0.17578473027227695]
```

[41]:
```
print(alphas_list)
```

```
[1.e-01 1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05]
```

Training

January 10, 2020

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow
import keras
from keras.applications.vgg16 import VGG16
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, AveragePooling2D
from keras.optimizers import SGD
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Dropout
from keras.models import Model
from multiprocessing import Pool

from keras.callbacks import ModelCheckpoint
from keras import backend
backend.set_image_data_format('channels_first')
```

```
Using TensorFlow backend.
```

0.0.1 Model configuration (transfer learning)

[3]:
```
loaded = np.load('compressed_train_test.npz')
x_train = loaded['x_train']
x_test = loaded['x_test']
y_train = loaded['y_train']
y_test = loaded['y_test']

print("Decompressing complete.")
```

```
Decompressing complete.
```

Definition

[ ]:
```
model = Sequential()
model.add(Convolution2D(4096, (6, 6), activation='relu', input_shape=(10, 12,␣
 ↪512), strides=(6, 6), name='input'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), activation='relu', strides=(1, 1),␣
 ↪name='conv_7'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), strides=(1, 1), name='conv_8'))
model.add(AveragePooling2D((2, 2), strides=(1, 1), name='add_pool'))

model.add(Flatten(name='flatten'))
model.add(Dense(3))
model.add(Activation("softmax"))
```

Compilation and Fitting

[7]:
```
opt = SGD(lr=1e-2)
#opt = keras.optimizers.Adam(lr=1.46e-3)
model.compile(loss="categorical_crossentropy", optimizer=opt,␣
 ↪metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=64, epochs=25,␣
 ↪validation_data=(x_test, y_test), shuffle=True, verbose=1)
```

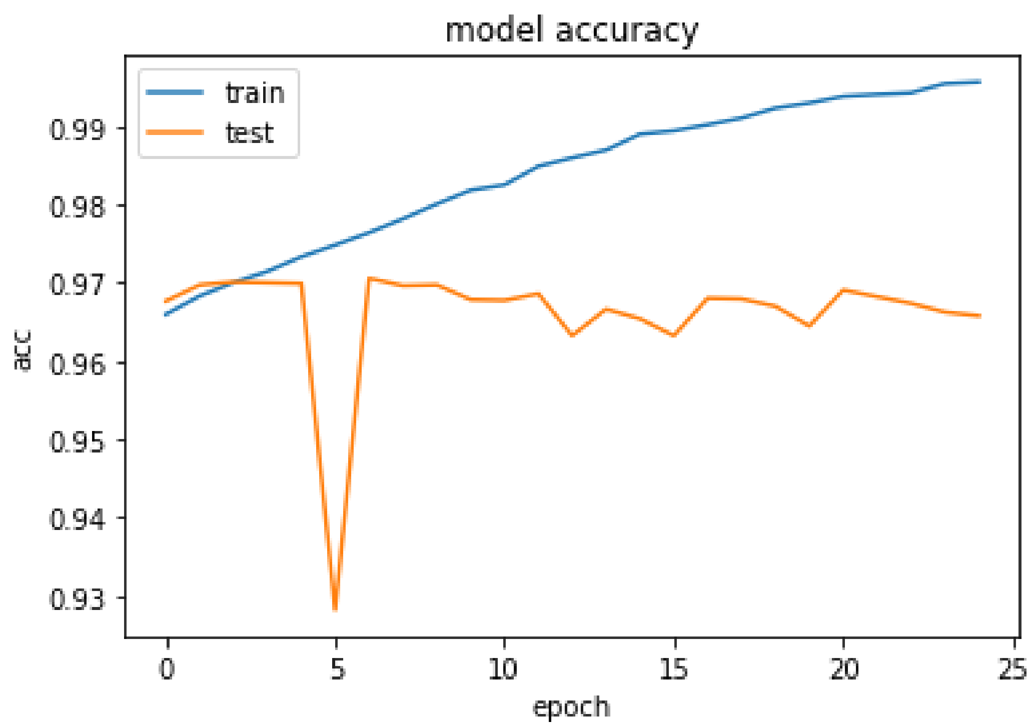
```
Train on 42508 samples, validate on 10628 samples
Epoch 1/25
42508/42508 [==============================] - 401s 9ms/step - loss: 0.1255 -
acc: 0.9659 - val_loss: 0.1169 - val_acc: 0.9676
Epoch 2/25
42508/42508 [==============================] - 415s 10ms/step - loss: 0.1097 -
acc: 0.9683 - val_loss: 0.1085 - val_acc: 0.9697
Epoch 3/25
42508/42508 [==============================] - 433s 10ms/step - loss: 0.0999 -
acc: 0.9700 - val_loss: 0.1147 - val_acc: 0.9701
Epoch 4/25
42508/42508 [==============================] - 329s 8ms/step - loss: 0.0919 -
acc: 0.9714 - val_loss: 0.1045 - val_acc: 0.9700
Epoch 5/25
42508/42508 [==============================] - 400s 9ms/step - loss: 0.0848 -
acc: 0.9733 - val_loss: 0.1037 - val_acc: 0.9699
Epoch 6/25
42508/42508 [==============================] - 386s 9ms/step - loss: 0.0776 -
acc: 0.9748 - val_loss: 0.2190 - val_acc: 0.9282
Epoch 7/25
42508/42508 [==============================] - 417s 10ms/step - loss: 0.0724 -
acc: 0.9764 - val_loss: 0.1066 - val_acc: 0.9705
Epoch 8/25
42508/42508 [==============================] - 527s 12ms/step - loss: 0.0655 -
```

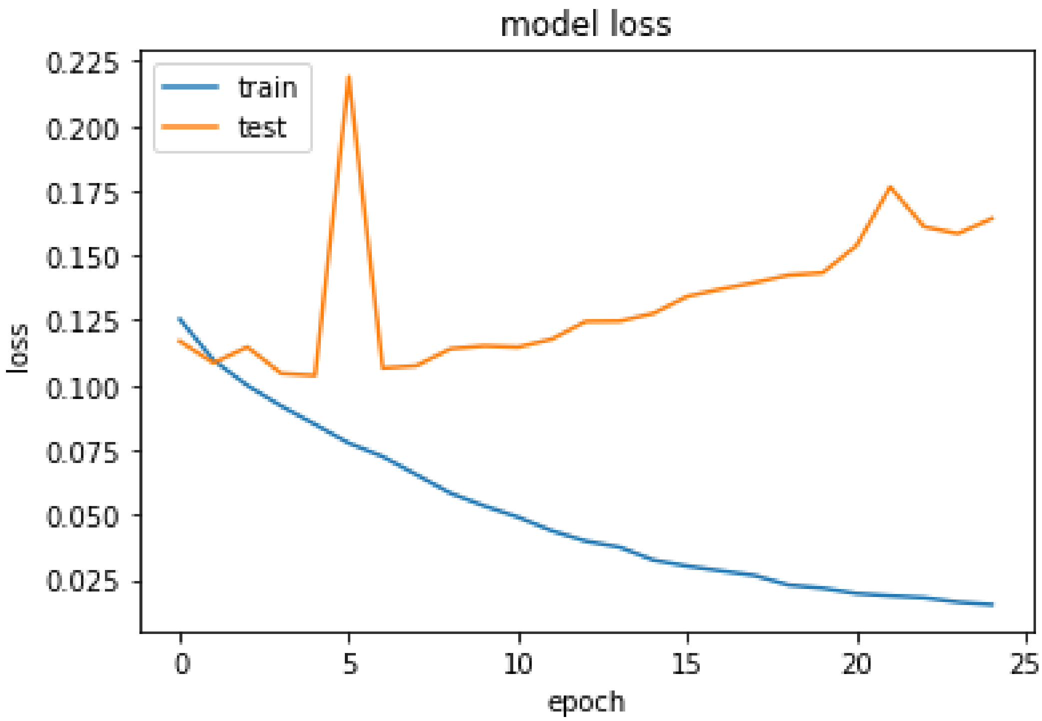
```
acc: 0.9781 - val_loss: 0.1075 - val_acc: 0.9696
Epoch 9/25
42508/42508 [==============================] - 297s 7ms/step - loss: 0.0583 -
acc: 0.9800 - val_loss: 0.1140 - val_acc: 0.9697
Epoch 10/25
42508/42508 [==============================] - 282s 7ms/step - loss: 0.0534 -
acc: 0.9818 - val_loss: 0.1151 - val_acc: 0.9678
Epoch 11/25
42508/42508 [==============================] - 277s 7ms/step - loss: 0.0491 -
acc: 0.9825 - val_loss: 0.1146 - val_acc: 0.9677
Epoch 12/25
42508/42508 [==============================] - 336s 8ms/step - loss: 0.0438 -
acc: 0.9848 - val_loss: 0.1177 - val_acc: 0.9686
Epoch 13/25
42508/42508 [==============================] - 328s 8ms/step - loss: 0.0398 -
acc: 0.9859 - val_loss: 0.1246 - val_acc: 0.9632
Epoch 14/25
42508/42508 [==============================] - 310s 7ms/step - loss: 0.0375 -
acc: 0.9869 - val_loss: 0.1247 - val_acc: 0.9666
Epoch 15/25
42508/42508 [==============================] - 343s 8ms/step - loss: 0.0325 -
acc: 0.9890 - val_loss: 0.1277 - val_acc: 0.9654
Epoch 16/25
42508/42508 [==============================] - 325s 8ms/step - loss: 0.0302 -
acc: 0.9894 - val_loss: 0.1342 - val_acc: 0.9632
Epoch 17/25
42508/42508 [==============================] - 281s 7ms/step - loss: 0.0283 -
acc: 0.9902 - val_loss: 0.1371 - val_acc: 0.9680
Epoch 18/25
42508/42508 [==============================] - 406s 10ms/step - loss: 0.0266 -
acc: 0.9910 - val_loss: 0.1396 - val_acc: 0.9679
Epoch 19/25
42508/42508 [==============================] - 397s 9ms/step - loss: 0.0229 -
acc: 0.9923 - val_loss: 0.1424 - val_acc: 0.9670
Epoch 20/25
42508/42508 [==============================] - 347s 8ms/step - loss: 0.0217 -
acc: 0.9929 - val_loss: 0.1433 - val_acc: 0.9644
Epoch 21/25
42508/42508 [==============================] - 353s 8ms/step - loss: 0.0196 -
acc: 0.9938 - val_loss: 0.1538 - val_acc: 0.9690
Epoch 22/25
42508/42508 [==============================] - 310s 7ms/step - loss: 0.0187 -
acc: 0.9940 - val_loss: 0.1764 - val_acc: 0.9682
Epoch 23/25
42508/42508 [==============================] - 313s 7ms/step - loss: 0.0180 -
acc: 0.9942 - val_loss: 0.1611 - val_acc: 0.9674
Epoch 24/25
42508/42508 [==============================] - 220s 5ms/step - loss: 0.0163 -
```

```
acc: 0.9954 - val_loss: 0.1584 - val_acc: 0.9662
Epoch 25/25
42508/42508 [==============================] - 248s 6ms/step - loss: 0.0153 -
acc: 0.9956 - val_loss: 0.1642 - val_acc: 0.9658
```

Plotting accuracies and losses

[8]:
```python
# Summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





Evaluating

[9]:
```
scores = model.evaluate(x_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
acc: 96.58%
```

[10]:
```
model.save("transfer_learning_model_25_epochs.h5")
print("Saved model to disk.")
```

```
Saved model to disk.
```

Wealth Predictions (transfer learning, CNN)

January 10, 2020

Deriving R^2 value when predicting asset wealth (CNN)

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.decomposition import PCA
```

[4]:
```
data_all = np.loadtxt('google_image_features_cnn.csv')
print("Features loaded.")
```

```
Features loaded.
```

[ ]:
```
x = data_all[:, :-1]
y = data_all[:, -1]
pca = PCA(n_components=1)
pca.fit(x)
proj = pca.transform(x)
```

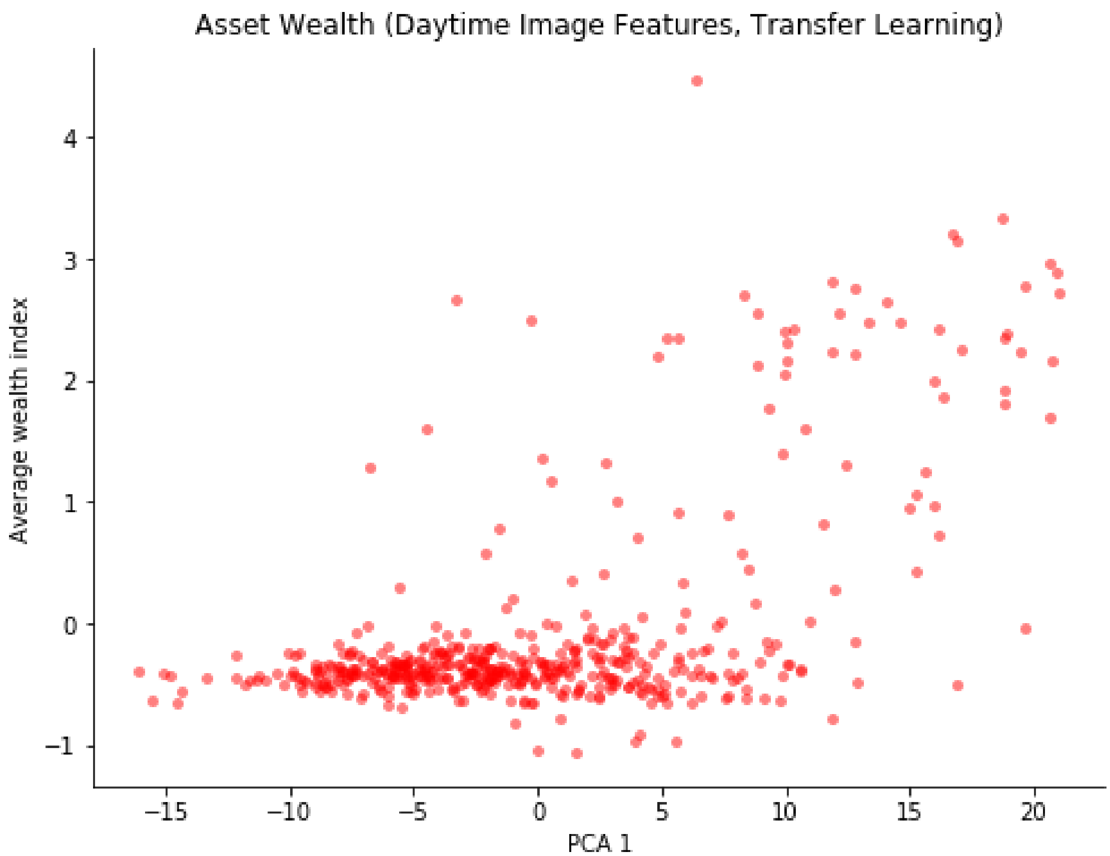
[27]:
```
print(len(y))
```

```
492
```

[21]:
```
clusters = pd.read_csv('rwanda_cluster_avg_asset_2010.csv')
wealth_known = clusters['wlthindf']
```

```
0     -0.531405
1     -0.409830
2     -0.478115
3     -0.435960
4     -0.449480
         …
487   -0.524310
488   -0.388025
489    0.328090
490   -0.278550
491   -0.310160
Name: wlthindf, Length: 492, dtype: float64
```

```
[30]: %matplotlib inline
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(proj[:,0], y, 'o', c='red', markersize=5, markeredgecolor='none',␣
 ↪alpha=0.5)
plt.xlabel('PCA 1')
plt.ylabel('Average wealth index')
plt.title('Asset Wealth (Daytime Image Features, Transfer Learning)')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')
```




```
[44]: predictions = []
observations = []

#alphas_list = np.logspace(-1, 5, 7)
alphas_list = [1.e+03]

final = []
for alpha in alphas_list:
```

```
    kf = KFold(n_splits=2, shuffle=True)
    scores = []
    for train_index, test_index in kf.split(data_all):
        reg = Ridge(alpha=alpha)
        train = data_all[train_index]
        test = data_all[test_index]
        train_x = train[:, :-1]
        train_y = train[:, -1]
        test_x = test[:, :-1]
        test_y = test[:, -1]
        # Reduce dimensions to avoid overfitting
        pca = PCA(n_components=100)
        pca.fit(train_x)
        train_x = pca.transform(train_x)
        test_x = pca.transform(test_x)
        reg.fit(train_x, train_y)

        prediction = reg.predict(test_x)
        predictions.append(prediction)
        observations.append(test_y)

        s = reg.score(test_x, test_y)
        scores.append(s)
    final.append(np.mean(scores))

print('R^2 of the best model: {:.3f}'.format(np.max(final)))
```

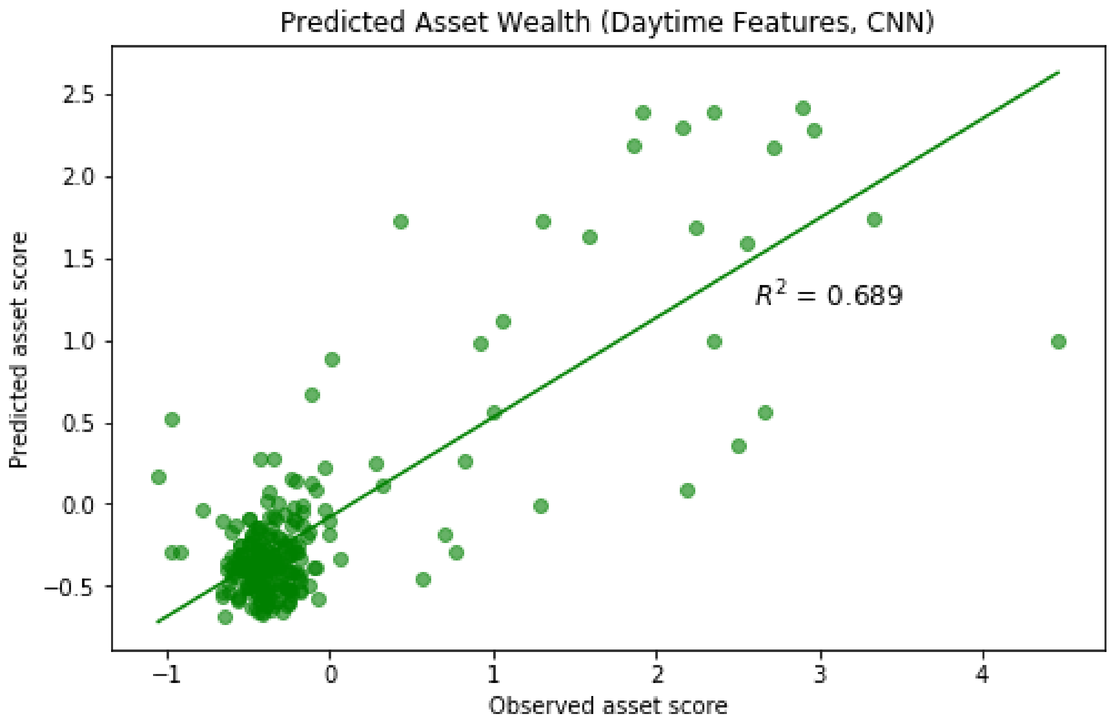
```
R^2 of the best model: 0.689
```

[45]:
```
print(alphas_list)
print(final)
```

```
[1000.0]
[0.6889992596548513]
```

[53]:
```
star_prediction = predictions[0]
star_observation = observations[0]

plt.figure(figsize=(8,5))
plt.scatter(star_observation, star_prediction, alpha=0.6, color='green')
plt.plot(np.unique(star_observation), np.poly1d(np.polyfit(star_observation,␣
 ↪star_prediction, 1))(np.unique(star_observation)), color='green')
plt.text(2.6, 1.2, '$R^2$ = 0.689', size=12)
plt.xlabel('Observed asset score')
plt.ylabel('Predicted asset score')
plt.title('Predicted Asset Wealth (Daytime Features, CNN)')
```

[53]: Text(0.5, 1.0, 'Predicted Asset Wealth (Daytime Features, CNN)')



Deriving final R^2 value when predicting asset wealth (transfer learning)

[4]:
```
data_all = np.loadtxt('google_image_features_cnn_retrain.csv')
print("Features loaded.")
```

Features loaded.

[4]:
```
predictions = []
observations = []
alphas_list = np.logspace(-1, 5, 7)

final = []
for alpha in alphas_list:
    kf = KFold(n_splits=10, shuffle=True)
    scores = []
    for train_index, test_index in kf.split(data_all):
        reg = Ridge(alpha=alpha)
        train = data_all[train_index]
        test = data_all[test_index]
        train_x = train[:, :-1]
        train_y = train[:, -1]
```

```
        test_x = test[:, :-1]
        test_y = test[:, -1]
        # Reduce dimensions to avoid overfitting
        pca = PCA(n_components=100)
        pca.fit(train_x)
        train_x = pca.transform(train_x)
        test_x = pca.transform(test_x)
        reg.fit(train_x, train_y)

        prediction = reg.predict(test_x)
        predictions.append(prediction)
        observations.append(test_y)

        s = reg.score(test_x, test_y)
        scores.append(s)
    final.append(np.mean(scores))

print('R^2 of the best model: {:.3f}'.format(np.max(final)))
```

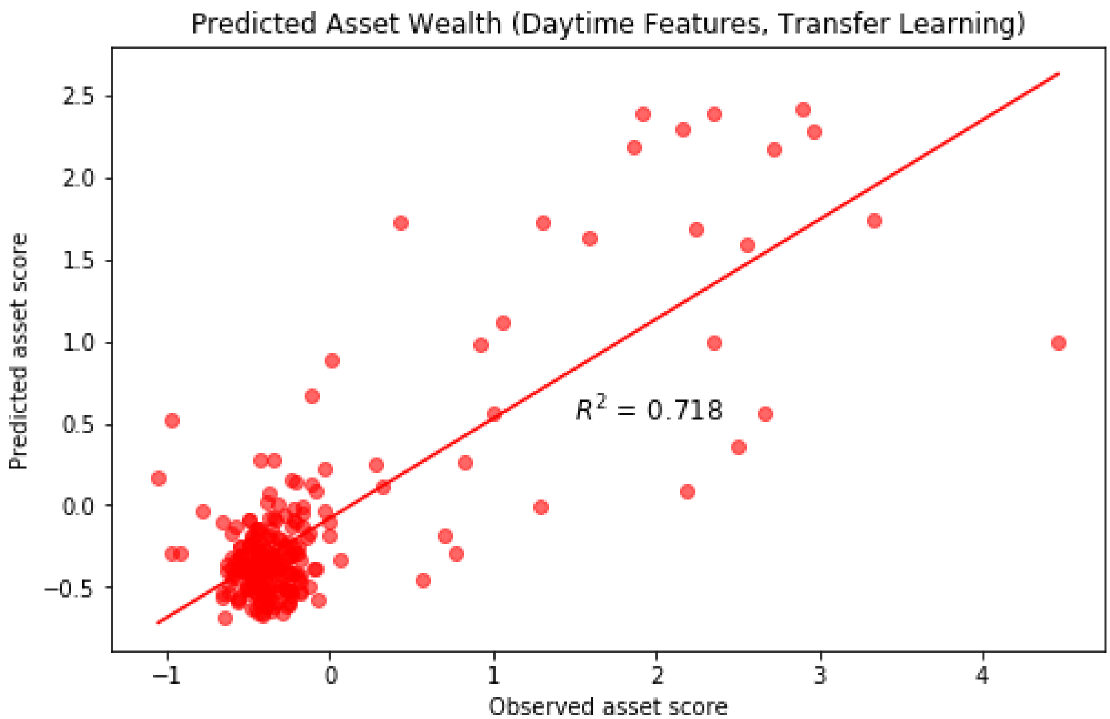
```
R^2 of the best model: 0.718
```

[11]:
```
print(alphas_list)
print(final)
```

```
[1.e-01 1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05]
[0.6571728522240462, 0.709800758403902, 0.6875553271555113, 0.711863772741731,
0.7183663582635443, 0.5483857781413544, 0.17578473027227695]
```

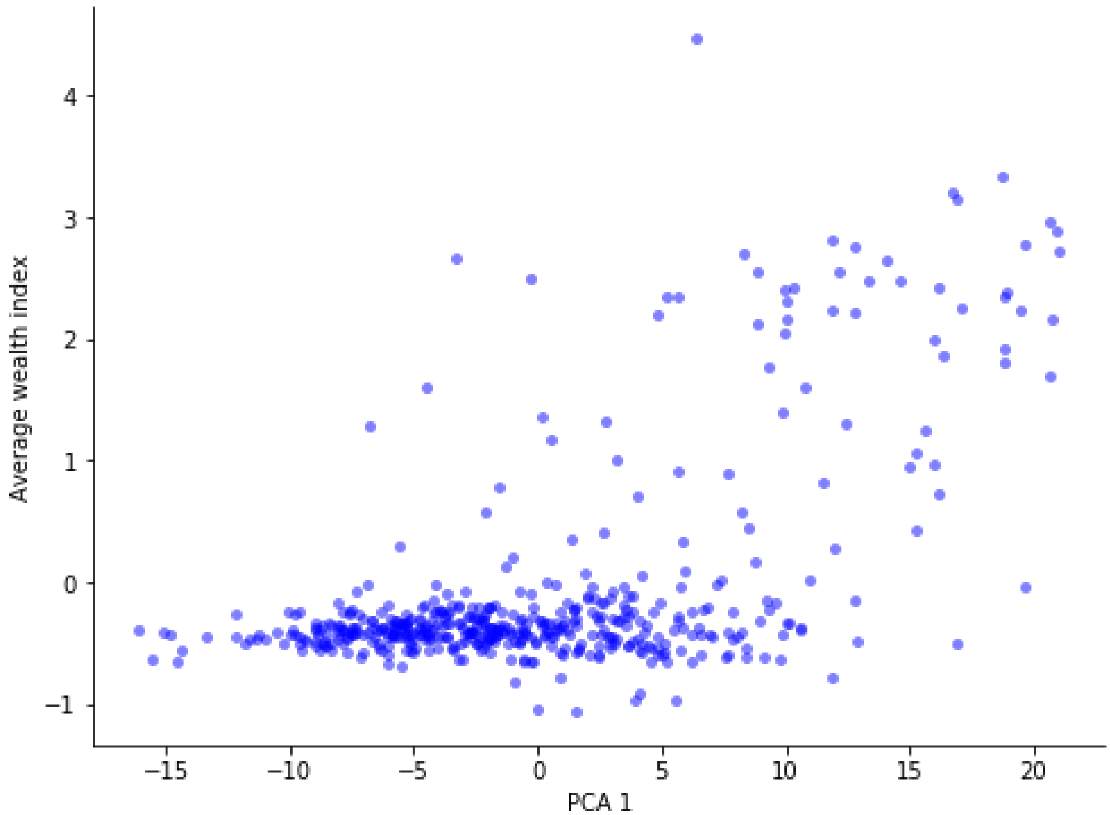
[46]:
```
star_prediction = predictions[4]
star_observation = observations[4]

plt.figure(figsize=(8,5))
plt.scatter(star_observation, star_prediction, alpha=0.6, color='red')
plt.plot(np.unique(star_observation), np.poly1d(np.polyfit(star_observation,␣
 ↪star_prediction, 1))(np.unique(star_observation)), color='red')
plt.text(1.5, 0.5, '$R^2$ = 0.718', size=12)
plt.xlabel('Observed asset score')
plt.ylabel('Predicted asset score')
plt.title('Predicted Asset Wealth (Daytime Features, Transfer Learning)')
```

[46]:
```
Text(0.5, 1.0, 'Predicted Asset Wealth (Daytime Features, Transfer Learning)')
```



Searching for optimal number of PCA components

[16]:
```
# For 'google_image_features_cnn_retrain.csv'
x = data_all[:, :-1]
y = data_all[:, -1]
pca = PCA(n_components=2)
pca.fit(x)
proj = pca.transform(x)

fig, ax = plt.subplots(figsize=(8,6))
ax.plot(proj[:,0], y, 'o', c='blue', markersize=5, markeredgecolor='none',␣
 ↪alpha=0.5)
plt.xlabel('PCA 1')
plt.ylabel('Average wealth index')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')
```


Average wealth index
PCA 1
4
3
2
1
0
−1
−15
−10
−5
0
5
10
15
20