ee292fproject

June 2, 2021

1 EE 292F Project

William Meng
EE 292F: Image Processing of Fine Arts
June 2, 2021

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib qt
from tqdm import tqdm
from skimage import io, color
import cv2
import os
from sklearn.decomposition import PCA
from scipy import ndimage
import plotly.express as px
```

2 Function definitions

[2]:
```
# Based on tutorial: https://jdhao.github.io/2017/11/06/
↪resize-image-to-square-with-padding/
def make_square(img, desired_size=256, fill_color=[255, 255, 255]):
    if img.dtype != np.uint8:
        print(f'Converting to uint8...')
        img = (255*img).astype(np.uint8)

    scale_factor = desired_size/max(img.shape[0], img.shape[1])
    resized = cv2.resize(img, (int(scale_factor*img.shape[1]),␣
↪int(scale_factor*img.shape[0])))
    new_size = resized.shape

    delta_w = desired_size - new_size[1]
    delta_h = desired_size - new_size[0]
    top, bottom = delta_h//2, delta_h-(delta_h//2)
    left, right = delta_w//2, delta_w-(delta_w//2)
```

```
    out = cv2.copyMakeBorder(resized, top, bottom, left, right, cv2.
↪BORDER_CONSTANT, value=fill_color)
    return out

# use float for thresholding, but return uint8 image
def binarize(img, threshold=0.5, invert=True):
    if img.dtype == np.uint8:
        img = img/255.0 # convert to float64

    # Convert to grayscale
    if len(img.shape) >= 3:
        img = color.rgb2gray(img)

    # Threshold
    out = np.zeros_like(img)
    if invert: # detect dark characters
        mask = img < threshold
    else: # detect light characters
        mask = img > threshold

    out[mask] = 1
    return (255*out).astype(np.uint8)

def preprocess(img, desired_size=256, threshold=0.5, invert=True):
    if invert: # detect black character on white background
        fill_color = [255, 255, 255]
    else: # detect white character on black background
        fill_color = [0, 0, 0]

    img_square = make_square(img, desired_size=desired_size,␣
↪fill_color=fill_color)
    img_bin = binarize(img_square, threshold=threshold, invert=invert)
    return img_bin

def feature_analysis(img, n=4, trim_points=10, bins=20, verbose=False):
    # Find all contours
    contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.
↪CHAIN_APPROX_NONE)
    if verbose:
        print(f'# of contours: {len(contours)}')
        print('# of points in each contour:')
        for cnt in contours:
            print(f'\t{len(cnt)} points')

    # Remove contours with too few points
    contours_trimmed = [cnt for cnt in contours if len(cnt) > trim_points]
```

```python
    if verbose:
        print(f'Trimming contours with fewer than {trim_points} points...')
        print(f'# of remaining contours: {len(contours_trimmed)}')
        for cnt in contours_trimmed:
            print(f'\t{len(cnt)} points')

    # Create dashed contours by keeping every nth point
    assert(n>=2)
    contours_dashed = [cnt[1::n] for cnt in contours_trimmed]
    if verbose:
        print(f'Taking every {n}th point to get dashed contour...')
        for cnt in contours_dashed:
            print(f'\t{len(cnt)} points')

    # Find angles between adjacent points in the contour
    thetaseq = []
    for i, cnt in enumerate(contours_dashed):
        for j, point in enumerate(cnt):
            if j == 0:
                prevx, prevy = point[0]
            else:
                x, y = point[0]
                thetaseq.append(np.arctan2(y-prevy, x-prevx))
                prevx = x
                prevy = y

    dthetaseq = np.diff(thetaseq)
    hist_theta = np.histogram(thetaseq, bins=bins, range=(-np.pi, np.pi),␣
↪density=True)
    hist_dtheta =  np.histogram(dthetaseq, bins=bins, range=(-np.pi, np.pi),␣
↪density=True)
    density_theta, theta = hist_theta
    density_dtheta, dtheta = hist_dtheta

    if verbose:
        print(f'# thetas: {len(thetaseq)}')
        print(f'max theta: {max(thetaseq)}')
        print(f'min theta: {min(thetaseq)}')
        print(f'len(thetaseq): {len(thetaseq)}')
        print(f'dthetaseq.shape: {dthetaseq.shape}')
        print(f'density_theta.shape: {density_theta.shape}')
        print(f'density_dtheta.shape: {density_dtheta.shape}')

    rows, cols = img.shape
    X, Y = np.meshgrid(np.linspace(0, 1, cols), np.linspace(0, 1, rows))
    assert(X.shape == Y.shape == img.shape)
    M = np.sum(img)
```

```
    M_norm = M/(255*rows*cols)
    EX = np.sum(X*img)/M
    EY = np.sum(Y*img)/M
    DX = np.sum(X**2 * img)/M - EX**2
    DY = np.sum(Y**2 * img)/M - EY**2
    covXY = np.sum(X*Y*img)/M - EX*EY

    features = np.hstack((density_theta, density_dtheta, M_norm, EX, EY, DX,␣
↪DY, covXY))
    #features = np.hstack((density_theta, density_dtheta, M_norm, DX, DY,␣
↪covXY))
    if verbose:
        print(f'features.shape: {features.shape}')
        print(f'M = {M}')
        print(f'M_norm = {M_norm}')
        print(f'EX = {EX}')
        print(f'EY = {EY}')
        print(f'DX = {DX}')
        print(f'DY = {DY}')
        print(f'covXY = {covXY}')
    return features

def plot_angles(hist, ax=None, title='Distribution of angles', label=''):
    r, theta = hist
    bins = len(r)
    #theta += np.pi/bins
    r = np.append(r, r[0]) # append 0th element to end cuz -pi and pi are the␣
↪same angle

    if ax is None:
        fig, ax = plt.subplots(subplot_kw={'projection': 'polar'}, figsize=(4,␣
↪4), dpi=300)

    ax.plot(theta, r, label=label)
    ax.grid(True)

    ax.set_title(title, va='bottom')
    plt.tight_layout()

def plot_hists(X, title='', filename='hists.png'):
    bins = 20
    theta = np.linspace(-np.pi, np.pi, bins+1)
    X_theta = X[0:bins]
    X_dtheta = X[bins:2*bins]

    fig = plt.figure(figsize=(8, 4))
    ax1 = plt.subplot(121, projection='polar')
```

```
    ax2 = plt.subplot(122, projection='polar')
    plot_angles([X_theta, theta], ax=ax1, title='Contour Angles')
    plot_angles([X_dtheta, theta], ax=ax2, title='Contour Curvatures')
    plt.suptitle(title, fontsize='xx-large')
    plt.tight_layout()
    plt.savefig(filename)

def feature_analysis_in_path(path='', bins=20, invert=True, verbose=False):
    filenames = os.listdir(path)
    filenames.sort()
    if verbose:
        print(filenames)

    num_features = 2*bins + 6
    #num_features = 2*bins + 4
    X = np.zeros((num_features, len(filenames)))
    for i, filename in enumerate(filenames):
        img = io.imread(path + filename)
        features = feature_analysis(preprocess(img, invert=invert), bins=bins)
        X[:, i] = features.T

    return X
```

3 PCA with 46 features

[3]:
```
paths = ['Extracted Characters/Mi Fu - Poem Written in a Boat on the Wu River/',
         'Extracted Characters/Emperor Huizong - Finches and bamboo/',
         'Extracted Characters/Su Shi - Inscription of Hanshi/',
         'Extracted Characters/WangXiZhi - On the Seventeenth Day/',
         'Extracted Characters/Mi Fu - On Cursive Calligraphy/']
label_list = ['Mi Fu (Wu River)',
              'Huizong',
              'Su Shi',
              'WangXiZhi',
              'Mi Fu (On Cursive Calligraphy)']
invert_list = [True,
               True,
               True,
               False,
               True]

# Construct labeled dataset of contour angle histograms
X_list = []
labels = []
for i, path in enumerate(paths):
    label = label_list[i]
```

```
    invert = invert_list[i]
    if invert:
        X = feature_analysis_in_path(path, invert=True)
    else:
        X = feature_analysis_in_path(path, invert=False)
    X_list.append(X)
    labels += [label] * X.shape[1]
    mu = np.mean(X, axis=1)
    plot_hists(mu, title=label, filename=f'Results/hists_{label.replace(" ",␣
↪"")}.png')

X_total =  np.vstack([X.T for X in X_list])

# Perform PCA
pca = PCA(n_components=2)
components = pca.fit_transform(X_total)
print(f'# of datapoints: {len(components)}')
print(f'len(labels) = {len(labels)}')
assert(len(labels) == len(components))
total_var = pca.explained_variance_ratio_.sum() * 100
print('Total explained variance = {:0.2f}%'.format(total_var))

# Plot results with Plotly
fig = px.scatter(components, x=0, y=1, color=labels)
fig.update_layout(
    title="PCA",
    xaxis_title="PC 1",
    yaxis_title="PC 2",
    legend_title="Class",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
fig.show()
fig.write_image('Results/PCA_Plotly_46features.png')
```

```
# of datapoints: 431
len(labels) = 431
Total explained variance = 56.90%
```

[4]:
```
# Compare 5 contour angle histograms
fig = plt.figure(figsize=(4, 22))
ax1 = plt.subplot(511, projection='polar')
ax2 = plt.subplot(512, projection='polar')
ax3 = plt.subplot(513, projection='polar')
```

```
ax4 = plt.subplot(514, projection='polar')
ax5 = plt.subplot(515, projection='polar')
axes = [ax1, ax2, ax3, ax4, ax5]

bins = 20
theta = np.linspace(-np.pi, np.pi, bins+1)
for i, X in enumerate(X_list):
    label = label_list[i]
    mu = np.mean(X, axis=1)
    plot_angles([mu[0:bins], theta], ax=axes[i], title=label)
    #plot_angles([mu[bins:2*bins], theta], ax=axes[i], title=label) # curvature
plt.suptitle('Contour Angle Statistics', fontsize='xx-large')
plt.tight_layout()
plt.savefig('Results/ContourAngleStatistics.png')
```

[5]:
```
# Compare 5 curvature histograms
fig = plt.figure(figsize=(4, 22))
ax1 = plt.subplot(511, projection='polar')
ax2 = plt.subplot(512, projection='polar')
ax3 = plt.subplot(513, projection='polar')
ax4 = plt.subplot(514, projection='polar')
ax5 = plt.subplot(515, projection='polar')
axes = [ax1, ax2, ax3, ax4, ax5]

bins = 20
theta = np.linspace(-np.pi, np.pi, bins+1)
for i, X in enumerate(X_list):
    label = label_list[i]
    mu = np.mean(X, axis=1)
    plot_angles([mu[bins:2*bins], theta], ax=axes[i], title=label) # curvature
plt.suptitle('Curvature Statistics', fontsize='xx-large')
plt.tight_layout()
plt.savefig('Results/CurvatureStatistics.png')
```

[6]:
```
# Show the Principal Components
pca.fit(X_total)
PC1, PC2 = pca.components_
print(f'PC 1 = {PC1}')
print(f'PC 2 = {PC2}')

plt.figure()
plt.plot(PC1, label='PC 1')
plt.plot(PC2, label='PC 2')
plt.legend()
plt.title('Principal Components')
plt.savefig('Results/PrincipalComponents.png')
```

```python
bins = 20
theta = np.linspace(-np.pi, np.pi, bins+1)

fig = plt.figure(figsize=(8, 4))
ax1 = plt.subplot(121, projection='polar')
ax2 = plt.subplot(122, projection='polar')
plot_angles([PC1[0:bins], theta], ax=ax1, title='Contour Angles')
plot_angles([PC2[0:bins], theta], ax=ax1, title='Contour Angles')
plot_angles([PC1[bins:2*bins], theta], ax=ax2, title='Contour Curvatures')
plot_angles([PC2[bins:2*bins], theta], ax=ax2, title='Contour Curvatures')
plt.suptitle('Principal Components', fontsize='xx-large')
plt.legend()
plt.tight_layout()
plt.savefig('Results/PC_hists.png')
```

```
PC 1 = [-7.25767574e-03  2.36275509e-02  8.67508098e-03  4.09302599e-02
  7.95683404e-03 -1.05213301e-01  5.36869689e-03 -2.71217901e-02
  4.77057561e-02 -1.94217970e-02 -1.16822276e-02  3.24863164e-02
  4.02041480e-02  3.14566220e-02  1.22382389e-02 -1.06009212e-01
 -1.45449558e-02 -9.18060673e-03  5.41130087e-02 -4.33094654e-03
  2.01023805e-03  4.17500486e-03  7.73578600e-03  2.41875523e-02
  1.34309776e-02  4.27761630e-02  6.46863625e-02  1.45067264e-01
  2.23372777e-01 -2.25138774e-01 -8.64214509e-01  2.37648884e-01
  1.55996375e-01  6.51992135e-02  3.63392053e-02  3.23055081e-02
  1.90580454e-02  6.05412620e-03  4.32106553e-03  4.98873479e-03
 -3.07541951e-02  1.89393050e-02 -9.89687117e-03 -3.70624308e-03
 -3.22107451e-04  4.23934567e-03]
PC 2 = [ 1.05582947e-01  4.46742876e-02  2.17561244e-02 -3.67611964e-02
 -1.23902754e-01 -5.22935215e-01 -8.36530952e-02  7.67507075e-02
  1.28096270e-01  2.16530136e-01  2.90408446e-01  4.22493722e-02
  9.66834113e-03 -3.90228116e-02 -9.67506547e-02 -5.57168871e-01
 -7.44295844e-02  7.06226817e-02  1.14493679e-01  4.13791189e-01
 -1.79664316e-03 -2.37742532e-03  5.77470620e-04  8.90403714e-04
  3.50473945e-04 -6.47410660e-03 -4.55137050e-03 -1.72659734e-02
 -2.88626291e-03  5.88496112e-02  9.06789617e-02 -4.05621316e-02
 -3.62783565e-02 -1.76425640e-02 -5.17802858e-03 -6.99191107e-03
 -6.57307243e-04 -2.82222484e-03 -2.76173321e-03 -3.10088220e-03
 -1.61498516e-02 -4.66897957e-03  4.35186681e-02 -4.40112165e-03
 -2.28709592e-03  6.63261791e-04]
```

```
No handles with labels found to put in legend.
```

4 Classification

First, split labeled data into training and test sets

[7]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_total, labels,␣
 ↪test_size=0.33, random_state=42)
print(f'X_total.shape: {X_total.shape}')
print(f'X_train.shape: {X_train.shape}')
print(f'X_test.shape: {X_test.shape}')
print(f'len(y_train): {len(y_train)}')
print(f'len(y_test): {len(y_test)}')
```

```
X_total.shape: (431, 46)
X_train.shape: (288, 46)
X_test.shape: (143, 46)
len(y_train): 288
len(y_test): 143
```

Now let's run a bunch of classifiers based on this tutorial: https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn

[8]:
```python
from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
 ↪GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="rbf", C=0.025, probability=True),
    NuSVC(probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    #QuadraticDiscriminantAnalysis(),
]

# Logging for Visual Comparison
log_cols=["Classifier", "Accuracy", "Log Loss"]
log = pd.DataFrame(columns=log_cols)

for clf in classifiers:
    clf.fit(X_train, y_train)
    name = clf.__class__.__name__
```

```python
    print("="*30)
    print(name)

    print('****Results****')
    train_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, train_predictions)
    print("Accuracy: {:.4%}".format(acc))

    train_predictions = clf.predict_proba(X_test)
    ll = log_loss(y_test, train_predictions)
    print("Log Loss: {}".format(ll))

    log_entry = pd.DataFrame([[name, acc*100, ll]], columns=log_cols)
    log = log.append(log_entry)

print("="*30)
```

```
==============================
KNeighborsClassifier
****Results****
Accuracy: 53.8462%
Log Loss: 7.033855174800827
==============================
SVC
****Results****
Accuracy: 22.3776%
Log Loss: 1.3117426374590893
==============================
NuSVC
****Results****
Accuracy: 61.5385%
Log Loss: 1.0261542999381201
==============================
DecisionTreeClassifier
****Results****
Accuracy: 48.9510%
Log Loss: 17.63168305474462
==============================
RandomForestClassifier
****Results****
Accuracy: 60.8392%
Log Loss: 1.028528459753278
==============================
AdaBoostClassifier
****Results****
Accuracy: 32.1678%
```

```
Log Loss: 1.7390233506540531
==============================
GradientBoostingClassifier
****Results****
Accuracy: 60.1399%
Log Loss: 1.1624200670033358
==============================
GaussianNB
****Results****
Accuracy: 61.5385%
Log Loss: 2.5987213761799137
==============================
LinearDiscriminantAnalysis
****Results****
Accuracy: 68.5315%
Log Loss: 1.0550883887981337
==============================
```

[9]:
```python
plt.figure(figsize=(8, 4))
sns.set_color_codes("muted")
sns.barplot(x='Accuracy', y='Classifier', data=log, color="b")

plt.xlabel('Accuracy %')
plt.title('Classifier Accuracy')
plt.xlim(0, 100)
plt.tight_layout()
plt.savefig('Results/ClassifierAccuracy.png')
```

[10]:
```python
plt.figure(figsize=(8, 4))
sns.set_color_codes("muted")
sns.barplot(x='Log Loss', y='Classifier', data=log, color="g")

plt.xlabel('Log Loss')
plt.title('Classifier Log Loss')
plt.xlim(0, 100)
plt.tight_layout()
plt.savefig('Results/ClassifierLogLoss.png')
```

[ ]:
```python

```