# Full Pipeline Scoring Issues (on Tileset7) - Aug 2017

Created: 24 Aug 2018
Last update: 24 Aug 2018

### This is mostly a copy of realxtals1-fullpipeline1, but with a 'break out' to figure out why scores deviate from prior work

---

## 1. Imports

In [82]:
```
# this will remove warnings messages
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

# import
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.pipeline import Pipeline
from sklearn.metrics import silhouette_score

import imgutils
```

In [83]:
```
# Re-run this cell if you altered imgutils
import importlib
importlib.reload(imgutils)
```

Out[83]:
```
<module 'imgutils' from 'C:\\JADS\\SW\\Grad Proj\\realxtals1\\sources\\imgutils.py'>
```

---

## 2. Data Definitions & Feature Specification

In [84]:
```
# Data:
datafolder = '../data/Crystals_Apr_12/Tileset7'
n_tiles_x = 3  # mostly for visualization
n_tiles_y = 2


# Features to use:
#feature_funcs = [imgutils.img_mean, imgutils.img_std, imgutils.img_median,
#                  imgutils.img_mode,
#                  imgutils.img_kurtosis, imgutils.img_skewness]
feature_funcs = [imgutils.img_std, imgutils.img_relstd, imgutils.img_mean,
                 imgutils.img_skewness,  imgutils.img_kurtosis, imgutils.img_mode]
feature_names = imgutils.stat_names(feature_funcs)

# Size of the grid, specified as number of slices per image in x and y direction:
n_rows = 4
n_cols = n_rows
```

---

## 3. Import Data & Extract Features

In [85]:
```
# image import:
print("Scanning for images in '{}'...".format(datafolder))
df_imgfiles = imgutils.scanimgdir(datafolder, '.tif')
imgfiles = list(df_imgfiles['filename'])
print("# of images: {} \n".format(len(imgfiles)))

# feature extraction:
print("Feature extraction...")
print("- Slicing up images in {} x {} patches. ".format(n_rows, n_cols))
print("- Extract statistics from each slice: {} ".format(', '.join(feature_names)))
print("...working...", end='\r')
df = imgutils.slicestats(imgfiles, n_rows, n_cols, feature_funcs)
print("# slices extracted: ", len(df))
```

```
Scanning for images in '../data/Crystals_Apr_12/Tileset7'...
# of images: 6

Feature extraction...
- Slicing up images in 4 x 4 patches.
- Extract statistics from each slice: img_std, img_relstd, img_mean, img_skewness, img_kurtosis, img_mode
# slices extracted:  96
```

# 4. Machine Learning Pipeline

## Hyper parameters

In [86]:
```
# data hyper-parameters
n_clusters = 3
n_important_features = len(feature_names)

# algorithm hyper-parameters:
kmeans_n_init = 10
pca_n_components = None   # i.e. all
```

In [87]:
```
def run_ml_pipeline(X, ml_name, ml_algorithm, standardize=True, use_pca=True):
    global pca_n_components

    # Setup algorithmic pipeline, including standardization
    pipeline = Pipeline([(ml_name, ml_algorithm)])

    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pipeline.steps.insert(0,('pca', PCA(n_components=pca_n_components)))
    if (standardize):
        pipeline.steps.insert(0, ('scaling_{0}'.format(ml_name), StandardScaler()))

    # run the pipelines
    y = pipeline.fit_predict(X) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_ml_pipelines(df_data, feature_cols, n_clust = n_clusters, standardize=True, use_pca=True):
    global pca_n_components, kmeans_n_init

    X = df_data.loc[:,feature_cols]

    # Setup ML clustering algorithms:
    kmeans = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    agglomerative =  AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')

    # run the pipelines
    print("Executing clustering pipelines...")
    score_kmeans, y_kmeans = run_ml_pipeline(X, 'kmeans', kmeans, standardize = standardize, use_pca = use_pca)
    score_hier, y_hier = run_ml_pipeline(X, 'hierarchical', agglomerative, standardize = standardize, use_pca = use_pca)
    print("Done\n")

    # collect data
    df_data['kmeans']=y_kmeans
    df_data['hierarchical']=y_hier

    # report results:
    print("\nClustering Scores:")
    print("K-means: ", score_kmeans)
    print("Hierarchical: ", score_hier)
```

In [109]:
```
# REGULAR PIPELINE
def run_kmeans_pipeline1(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    X = df_data.loc[:,feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    ml_name = 'kmeans1'
    pipeline = Pipeline([(ml_name, ml_algorithm)])

    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pipeline.steps.insert(0,('pca', PCA(n_components=pca_n_components)))
    if (standardize):
        pipeline.steps.insert(0, ('scaling_{0}'.format(ml_name), StandardScaler()))

    #print(pipeline.steps)

    # run the pipelines
    y = pipeline.fit_predict(X) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_hierarchical_pipeline1(df_data, feature_cols, n_clust=n_clusters, standardize=True, use_pca=True):
    global pca_n_components


    X = df_data.loc[:,feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')
    ml_name = 'hier1'
    pipeline = Pipeline([(ml_name, ml_algorithm)])

    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pipeline.steps.insert(0,('pca', PCA(n_components=pca_n_components)))
    if (standardize):
        pipeline.steps.insert(0, ('scaling_{0}'.format(ml_name), StandardScaler()))

    #print(pipeline.steps)

    # run the pipelines
    y = pipeline.fit_predict(X) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [110]:
```
# ORIGINAL step-by-step
def run_kmeans_pipeline2(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    #print("Settings: ", standardize, use_pca)

    X = []
    if (standardize):
        imgutils.normalize(df_data, feature_cols)
        norm_feature_cols = imgutils.normalized_names(feature_cols)
        X = df_data.loc[:,norm_feature_cols]
    else:
        X = df_data.loc[:,feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    ml_name = 'kmeans2'

    X_use = X
    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pca = PCA(n_components=pca_n_components)
        X_use = pca.fit_transform(X)

    # run the pipelines
    y = ml_algorithm.fit_predict(X_use) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_hierarchical_pipeline2(df_data, feature_cols, n_clust=n_clusters, standardize=True, use_pca=True):
    global pca_n_components

    #print("Settings: ", standardize, use_pca)

    X = []
    if (standardize):
        imgutils.normalize(df_data, feature_cols)
        norm_feature_cols = imgutils.normalized_names(feature_cols)
        X = df_data.loc[:,norm_feature_cols]
    else:
        X = df_data.loc[:,feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')
    ml_name = 'hier2'

    X_use = X
    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pca = PCA(n_components=pca_n_components)
        X_use = pca.fit_transform(X)

    # run the pipelines
    y = ml_algorithm.fit_predict(X_use) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [111]:
```python
# Pipeline differently composed
def run_kmeans_pipeline3(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    X = df_data.loc[:,feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    ml_name = 'kmeans3'
    pl_contents = []

    if (standardize):
        pl_contents.append(('scaling_{0}'.format(ml_name), StandardScaler()))
    if (use_pca):
        pl_contents.append(('pca_{0}'.format(ml_name), PCA(n_components=pca_n_components)))
    pl_contents.append((ml_name, ml_algorithm))

    pipeline = Pipeline(pl_contents)
    #print(pipeline.steps)

    # run the pipelines
    y = pipeline.fit_predict(X) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_hierarchical_pipeline3(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    X = df_data.loc[:,feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')
    ml_name = 'hier3'
    pl_contents = []

    if (standardize):
        pl_contents.append(('scaling_{0}'.format(ml_name), StandardScaler()))
    if (use_pca):
        pl_contents.append(('pca_{0}'.format(ml_name), PCA(n_components=pca_n_components)))
    pl_contents.append((ml_name, ml_algorithm))

    pipeline = Pipeline(pl_contents)
    #print(pipeline.steps)

    # run the pipelines
    y = pipeline.fit_predict(X) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [91]:
```
# Step-by-step but with StandardScaler
def run_kmeans_pipeline4(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    X = df_data.loc[:,feature_cols]
    X_use = X

    if (standardize):
        scaler = StandardScaler()
        X_use = scaler.fit_transform(X)

    if (use_pca):
        pca = PCA(n_components=pca_n_components)
        X_use = pca.fit_transform(X_use)

    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    ml_name = 'kmeans4'
    y = ml_algorithm.fit_predict(X_use)

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_hierarchical_pipeline4(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    X = df_data.loc[:,feature_cols]
    X_use = X

    if (standardize):
        scaler = StandardScaler()
        X_use = scaler.fit_transform(X)

    if (use_pca):
        pca = PCA(n_components=pca_n_components)
        X_use = pca.fit_transform(X_use)

    ml_algorithm = AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')
    ml_name = 'hier4'

    y = ml_algorithm.fit_predict(X_use)

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [112]:

```
# pipeline custom scaler
def run_kmeans_pipeline5(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    #print("Settings: ", standardize, use_pca)

    X = df_data.loc[:,feature_cols]

    X_use = X
    if (standardize):
        imgutils.normalize(df_data, feature_cols)
        norm_feature_cols = imgutils.normalized_names(feature_cols)
        X_use = df_data.loc[:,norm_feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    ml_name = 'kmeans5'
    pipeline = Pipeline([(ml_name, ml_algorithm)])

    if (use_pca):
        pipeline.steps.insert(0,('pca', PCA(n_components=pca_n_components)))

    #print(pipeline.steps)

    # run the pipelines
    y = pipeline.fit_predict(X_use) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_hierarchical_pipeline5(df_data, feature_cols, n_clust=n_clusters, standardize=True, use_pca=True):
    global pca_n_components

    #print("Settings: ", standardize, use_pca)

    X = df_data.loc[:,feature_cols]

    X_use = X
    if (standardize):
        imgutils.normalize(df_data, feature_cols)
        norm_feature_cols = imgutils.normalized_names(feature_cols)
        X_use = df_data.loc[:,norm_feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')
    ml_name = 'hier5'

    pipeline = Pipeline([(ml_name, ml_algorithm)])

    if (use_pca):
        pipeline.steps.insert(0,('pca', PCA(n_components=pca_n_components)))

    #print(pipeline.steps)

    # run the pipelines
    y = pipeline.fit_predict(X_use) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [113]:
```
# ORIGINAL ALT SCORE (step-by-step)
def run_kmeans_pipeline6(df_data, feature_cols, n_clust=n_clusters,standardize=True, use_pca=True):
    global pca_n_components

    #print("Settings: ", standardize, use_pca)

    X = df_data.loc[:,feature_cols]

    X_use = X
    if (standardize):
        imgutils.normalize(df_data, feature_cols)
        norm_feature_cols = imgutils.normalized_names(feature_cols)
        X_use = df_data.loc[:,norm_feature_cols]

    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = KMeans(algorithm='auto', n_clusters=n_clust, n_init=kmeans_n_init, init='k-means++')
    ml_name = 'kmeans6'

    if (use_pca):
        pca = PCA(n_components=pca_n_components)
        X_use = pca.fit_transform(X_use)

    # run the pipelines
    y = ml_algorithm.fit_predict(X_use) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y

def run_hierarchical_pipeline6(df_data, feature_cols, n_clust=n_clusters, standardize=True, use_pca=True):
    global pca_n_components

    #print("Settings: ", standardize, use_pca)

    X = df_data.loc[:,feature_cols]

    X_use = X
    if (standardize):
        imgutils.normalize(df_data, feature_cols)
        norm_feature_cols = imgutils.normalized_names(feature_cols)
        X_use = df_data.loc[:,norm_feature_cols]


    # Setup algorithmic pipeline, including standardization if enabled
    ml_algorithm = AgglomerativeClustering(n_clusters=n_clust, affinity='euclidean', linkage='complete')
    ml_name = 'hier6'

    # watch the order, pca should happen after scaling, but we insert at 0
    if (use_pca):
        pca = PCA(n_components=pca_n_components)
        X_use = pca.fit_transform(X_use)

    # run the pipelines
    y = ml_algorithm.fit_predict(X_use) # calls predict on last step to get the labels

    # report score:
    score = silhouette_score(X, y)

    return score, y
```

In [114]:
```
run_ml_pipelines(df, feature_names, standardize=True, use_pca=True)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.3834680571101721
Hierarchical:  0.7100243618056425
```

In [115]:
```
print("Pipeline-inserts:")
score_k_1, y_k_1 = run_kmeans_pipeline1(df, feature_names, standardize=True, use_pca=False)
score_h_1, y_h_1 = run_hierarchical_pipeline1(df, feature_names, standardize=True, use_pca=False)
print('K-means 1:', score_k_1)
print('Hier. 1:', score_h_1)
```

```
Pipeline-inserts:
K-means 1: 0.3834680571101721
Hier. 1: 0.7100243618056425
```

In [116]:
```
print("Pipeline-inserts PCA:")
score_k_1p, y_k_1p = run_kmeans_pipeline1(df, feature_names, standardize=True, use_pca=True)
score_h_1p, y_h_1p = run_hierarchical_pipeline1(df, feature_names, standardize=True, use_pca=True)
print('K-means PCA 1:', score_k_1p)
print('Hier. PCA 1:', score_h_1p)
```

```
Pipeline-inserts PCA:
K-means PCA 1: 0.3834680571101721
Hier. PCA 1: 0.7100243618056425
```

In [117]:
```
print("ORIGINAL (step-by-step):")
score_k_2, y_k_2 = run_kmeans_pipeline2(df, feature_names, standardize=True, use_pca=False)
score_h_2, y_h_2 = run_hierarchical_pipeline2(df, feature_names, standardize=True, use_pca=False)
print('K-means 2:', score_k_2)
print('Hier. 2:', score_h_2)
```

```
ORIGINAL (step-by-step):
K-means 2: 0.5272951661092291
Hier. 2: 0.5543189694833234
```

In [118]:
```
print("ORIGINAL PCA (step-by-step):")
score_k_2p, y_k_2p = run_kmeans_pipeline2(df, feature_names, standardize=True, use_pca=True)
score_h_2p, y_h_2p = run_hierarchical_pipeline2(df, feature_names, standardize=True, use_pca=True)
print('K-means PCA 2:', score_k_2p)
print('Hier. PCA 2:', score_h_2p)
```

```
ORIGINAL PCA (step-by-step):
K-means PCA 2: 0.5251916981567151
Hier. PCA 2: 0.5543189694833234
```

In [119]:
```
print("Alt. Pipeline:")
score_k_3, y_k_3 = run_kmeans_pipeline3(df, feature_names, standardize=True, use_pca=False)
score_h_3, y_h_3 = run_hierarchical_pipeline3(df, feature_names, standardize=True, use_pca=False)
print('K-means 3:', score_k_3)
print('Hier. 3:', score_h_3)
```

```
Alt. Pipeline:
K-means 3: 0.38346805711101721
Hier. 3: 0.7100243618056425
```

In [120]:
```
print("Alt. Pipeline PCA:")
score_k_3p, y_k_3p = run_kmeans_pipeline3(df, feature_names, standardize=True, use_pca=True)
score_h_3p, y_h_3p = run_hierarchical_pipeline3(df, feature_names, standardize=True, use_pca=True)
print('K-means 3 PCA:', score_k_3p)
print('Hier. 3 PCA:', score_h_3p)
```

```
Alt. Pipeline PCA:
K-means 3 PCA: 0.38346805711101721
Hier. 3 PCA: 0.7100243618056425
```

In [121]:
```
print("Alt Original (step-by-step, StandardScaler):")
score_k_4, y_k_4 = run_kmeans_pipeline4(df, feature_names, standardize=True, use_pca=False)
score_h_4, y_h_4 = run_hierarchical_pipeline4(df, feature_names, standardize=True, use_pca=False)
print('K-means 4:', score_k_4)
print('Hier. 4:', score_h_4)
```

```
Alt Original (step-by-step, StandardScaler):
K-means 4: 0.494192683227838
Hier. 4: 0.7100243618056425
```

In [122]:
```
print("Alt Original PCA(step-by-step, StandardScaler):")
score_k_4p, y_k_4p = run_kmeans_pipeline4(df, feature_names, standardize=True, use_pca=True)
score_h_4p, y_h_4p = run_hierarchical_pipeline4(df, feature_names, standardize=True, use_pca=True)
print('K-means 4 PCA:', score_k_4p)
print('Hier. 4 PCA:', score_h_4p)
```

```
Alt Original PCA(step-by-step, StandardScaler):
K-means 4 PCA: 0.38346805711101721
Hier. 4 PCA: 0.7100243618056425
```

In [123]:
```
print("Pipeline - custom scaling:")
score_k_5, y_k_5 = run_kmeans_pipeline5(df, feature_names, standardize=True, use_pca=False)
score_h_5, y_h_5 = run_hierarchical_pipeline5(df, feature_names, standardize=True, use_pca=False)
print('K-means 5:', score_k_5)
print('Hier. 5:', score_h_5)
```

```
Pipeline - custom scaling:
K-means 5: 0.38346805711101721
Hier. 5: 0.7100243618056425
```

In [124]:
```
print("Pipeline PCA - custom scaling:")
score_k_5p, y_k_5p = run_kmeans_pipeline5(df, feature_names, standardize=True, use_pca=True)
score_h_5p, y_h_5p = run_hierarchical_pipeline5(df, feature_names, standardize=True, use_pca=True)
print('K-means PCA 5:', score_k_5p)
print('Hier. PCA 5:', score_h_5p)
```

```
Pipeline PCA - custom scaling:
K-means PCA 5: 0.38346805711101721
Hier. PCA 5: 0.7100243618056425
```

## Summary of all scorings:

- K-means 1: 0.3732994909365581 pipeline
- K-means 2: 0.5251916981567151 step-by-step <--- WHAT IS GOING ON HERE?
- K-means 3: 0.3732994909365581 alt pipeline
- K-means 4: 0.3732994909365581 step-by-step StandardScaler
- K-means 5: 0.3732994909365581 pipeline custom scaler
- K-means PCA 1: 0.3732994909365581 pipeline
- K-means PCA 2: 0.5272951661092291 step-by-step <--- WHAT IS GOING ON HERE?
- K-means PCA 3: 0.3834680571101721 alt pipeline
- K-means PCA 4: 0.3732994909365581 step-by-step StandardScaler
- K-means PCA 5: 0.3834680571101721 pipeline custom scaler
- Hier. 1: 0.7100243618056425 pipeline
- Hier. 2: 0.5543189694833234 step-by-step <--- WHAT IS GOING ON HERE?
- Hier. 3: 0.7100243618056425 alt pipeline
- Hier. 4: 0.7100243618056425 step-by-step StandardScaler
- Hier. 5: 0.7100243618056425 pipeline custom scaler
- Hier. PCA 1: 0.7100243618056425 pipeline
- Hier. PCA 2: 0.5543189694833234 step-by-step <--- WHAT IS GOING ON HERE?
- Hier. PCA 3: 0.7100243618056425 alt pipeline
- Hier. PCA 4: 0.7100243618056425 step-by-step StandardScaler
- Hier. PCA 5: 0.7100243618056425 pipeline custom scaler

### It's not the standardscaler or pipeline, there is someing in the step-by-step impl.

(after close inspection, I found it. It's the score calculation)

In [144]:
```
print("ORIGINAL ALT SCORE (step-by-step):")
score_k_6, y_k_6 = run_kmeans_pipeline6(df, feature_names, standardize=True, use_pca=False)
score_h_6, y_h_6 = run_hierarchical_pipeline6(df, feature_names, standardize=True, use_pca=False)
print('K-means 6:', score_k_6)
print('Hier. 6:', score_h_6)
```

```
ORIGINAL ALT SCORE (step-by-step):
K-means 6: 0.3834680571101721
Hier. 6: 0.7100243618056425
```

In [145]:
```
print("ORIGINAL ALT SCORE PCA (step-by-step):")
score_k_6p, y_k_6p = run_kmeans_pipeline6(df, feature_names, standardize=True, use_pca=True)
score_h_6p, y_h_6p = run_hierarchical_pipeline6(df, feature_names, standardize=True, use_pca=True)
print('K-means PCA 6:', score_k_6p)
print('Hier. PCA 6:', score_h_6p)
```

```
ORIGINAL ALT SCORE PCA (step-by-step):
K-means PCA 6: 0.3732994909365581
Hier. PCA 6: 0.7100243618056425
```

This is indeed the same as all the other implementations.

### Issue solved! it was the scoring

(the original step-by-step used the normalized data to calculate the silouette score, while all other variant are using the unnormalized data).

Hence, running the pipeline without normalization will give a higher score (see next step), though that does not mean it's better. Needs visual inspection

In [146]:
```
run_ml_pipelines(df, feature_names, standardize=False, use_pca=False)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.600017787779939
Hierarchical:  0.6138055241918071
```

to make it the same as the 'step-by-step' outcome, I need to calc the score of the pipeline output with the normalized features (which are in the dataframe)

In [147]:
```
run_ml_pipelines(df, feature_names, standardize=True, use_pca=False)
norm_features = imgutils.normalized_names(feature_names)
x_base = df[norm_features]
print('\nRe-calculating scores...:')
print('Score k-means (norm): ', silhouette_score(x_base, df['kmeans']))
print('Score hier. (norm): ', silhouette_score(x_base, df['hierarchical']))
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.3834680571101721
Hierarchical:  0.7100243618056425

Re-calculating scores...:
Score k-means (norm):  0.5251916981567151
Score hier. (norm):  0.5543189694833234
```

**This is indeed (almost) identical to original step-by-step. Problem resolved!**

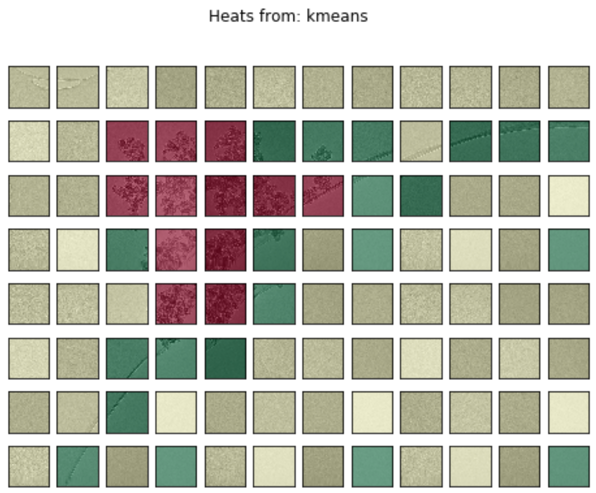
## 5. Visualize with and without normalization

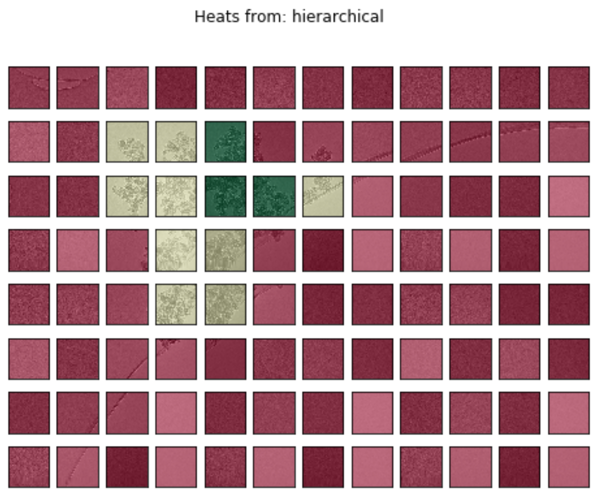
In [148]:
```
s = (8,6)
```

In [149]:
```
run_ml_pipelines(df, feature_names, standardize=True, use_pca=True)
print("WITH NORMALIZATION:")
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.3732994909365581
Hierarchical:  0.7100243618056425
WITH NORMALIZATION:
```



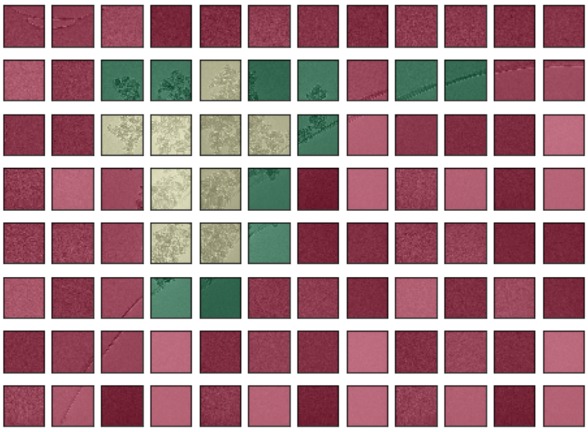



In [150]:
```
run_ml_pipelines(df, feature_names, standardize=False, use_pca=True)
print("NO NORMALIZATION:")
imgutils.show_large_heatmap(df, 'kmeans', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
imgutils.show_large_heatmap(df, 'hierarchical', imgfiles, n_rows=n_tiles_y, n_cols=n_tiles_x, fig_size=s)
```

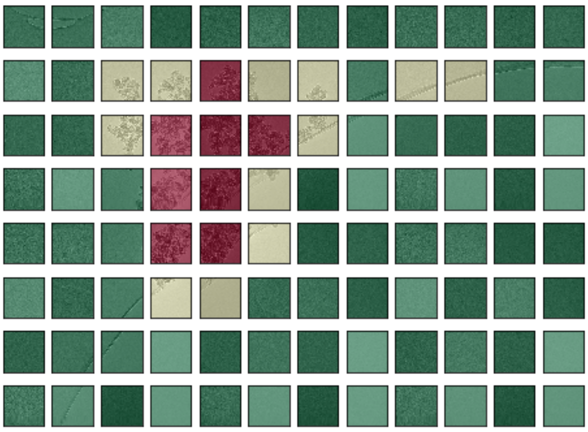
```
Executing clustering pipelines...
Done


Clustering Scores:
K-means:  0.600017787779939
Hierarchical:  0.6138055241918071
NO NORMALIZATION:
```

Heats from: kmeans



Heats from: hierarchical



### Looks like hierarchical works better without normalization, k-means with normalization

---

## 6. Conclusions & Next Steps

- Scoring issue is resolved!
- The difference is not coming from any software error
- It depends on how the score is calculated; using the unnormalized or normalized data as basis (though normalized data is used for the unsupervised learning)
- For this data, hierarchical clustering works better without normalization. ### Next Step: Back to the full pipeline development!