02-model-analysis

December 17, 2021

1 Imports

[1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import pandas as pd
import random
import time
import joblib
import os
from utils import get_dataset_files, extract_random_entries,␣
 ↪extract_first_entries, generate_pixel_columns, load_run,␣
 ↪extract_best_entries, render_single
from IPython.display import display, Image as IPImage
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC, NuSVC, SVC
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
 ↪QuadraticDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.kernel_ridge import KernelRidge
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from itertools import repeat
from sklearn.metrics import accuracy_score, classification_report
```

2 Data loading + generation

Commented out are a few alternate ways of loading the data. - Loading all classes, specific classes, or a certain number of classes at random - Loading all entries in a class, loading a certain number of random entries, loading a certain number of the first entries in a class, or loading a certain number of the most complex entries in a class

[2]:

```
load_existing_run = None

if load_existing_run is None:
    # num_cats = 10
    entries_per_cat = 2000
    image_gen_params = {
        'magnification': 4, # Higher values improve antialiasing, but uses more␣
↪memory during drawing
        'resolution': 32,
        'invert_color': True, # True = white on black
        'stroke_width_scale': 2 # What stroke width to use to trace the lines␣
↪in the drawing
    }

    # files = get_dataset_files()
    # files = random.sample(files, num_cats)
    # names = ['power outlet', 'pickup truck', 'castle']
    names = ['ambulance','bed','bench','bowtie','bread','castle','cell␣
↪phone','chair','church','coffee cup','crown','cruise␣
↪ship','cup','dishwasher','dresser',
        'eye','face','fan','fire␣
↪hydrant','fish','hammer','hat','helicopter','ice␣
↪cream','lantern','passport','pickup truck','pillow','power␣
↪outlet','sailboat',
        ␣
↪'sandwich','snowman','star','strawberry','suitcase','table','telephone','traffic␣
↪light','watermelon','wine glass']
    files = list(map(lambda n: f"./dataset/{n}.ndjson", names))
    df = extract_best_entries(files, entries_per_cat, recognized=True,␣
↪skip_first=200)
    # df = extract_random_entries(files, entries_per_cat, recognized=True)
    # df = extract_best_entries(files, entries_per_cat, recognized=True)

    print(f'Loaded {len(df)} entries from {files}')
    df = df.sample(len(df))
    print('Done shuffling dataset')
    df = generate_pixel_columns(df, **image_gen_params).reset_index(drop=True)
    print('Done generating pixel columns')

else:
    run = load_run(load_existing_run)
    df = run['data']
    num_cats = len(df['word'].value_counts())
    entries_per_cat = df['word'].value_counts()[df['word'].value_counts().
↪keys()[0]]
    image_gen_params = run['img_params']
```

```
Loaded 80000 entries from ['./dataset/ambulance.ndjson', './dataset/bed.ndjson',
'./dataset/bench.ndjson', './dataset/bowtie.ndjson', './dataset/bread.ndjson',
'./dataset/castle.ndjson', './dataset/cell phone.ndjson',
'./dataset/chair.ndjson', './dataset/church.ndjson', './dataset/coffee
cup.ndjson', './dataset/crown.ndjson', './dataset/cruise ship.ndjson',
'./dataset/cup.ndjson', './dataset/dishwasher.ndjson',
'./dataset/dresser.ndjson', './dataset/eye.ndjson', './dataset/face.ndjson',
'./dataset/fan.ndjson', './dataset/fire hydrant.ndjson',
'./dataset/fish.ndjson', './dataset/hammer.ndjson', './dataset/hat.ndjson',
'./dataset/helicopter.ndjson', './dataset/ice cream.ndjson',
'./dataset/lantern.ndjson', './dataset/passport.ndjson', './dataset/pickup
truck.ndjson', './dataset/pillow.ndjson', './dataset/power outlet.ndjson',
'./dataset/sailboat.ndjson', './dataset/sandwich.ndjson',
'./dataset/snowman.ndjson', './dataset/star.ndjson',
'./dataset/strawberry.ndjson', './dataset/suitcase.ndjson',
'./dataset/table.ndjson', './dataset/telephone.ndjson', './dataset/traffic
light.ndjson', './dataset/watermelon.ndjson', './dataset/wine glass.ndjson']
Done shuffling dataset
Done generating pixel columns
```

3 Data splitting, standardization, and dimensional reduction

[3]:
```
train_amt = int(len(df) * .8)

train = df[:train_amt]
test = df[train_amt:]

train = train.reset_index(drop=True)
test = test.reset_index(drop=True)

print(f'Train: {len(train)} entries, test: {len(test)} entries.')

pca_on = True

y = train['word'].to_numpy()
X = train.filter(regex='pixel.+').to_numpy()
print("Done generating features and target")

if pca_on:
    if load_existing_run is None:
        scaler = StandardScaler()
        X = scaler.fit_transform(X)
        pca = PCA(.85)
        X = pca.fit_transform(X)
        print(f'PCA & standardization done. Keeping {pca.n_components_}␣
↪features')
    else:
```

```
        scaler = run['scaler']
        pca = run['pca']
        X = scaler.transform(X)
        X = pca.transform(X)
        print('Applied scaler and PCA.')

save_to_disk = True

if save_to_disk:
    stamp = str(int(time.time()))
    folder = f'./runs/{stamp}/'
    if not os.path.exists(folder):
        os.makedirs(folder)
    pd.DataFrame.to_feather(df, folder + 'data')
    with open(folder + 'img_params', 'w') as f:
        f.writelines(str(image_gen_params))
    print('Done saving dataset to disk')
    if pca_on:
        joblib.dump(pca, folder + 'pca')
        joblib.dump(scaler, folder + 'scaler')
        print('Done saving PCA and scaler to disk')
```

```
Train: 64000 entries, test: 16000 entries.
Done generating features and target
PCA & standardization done. Keeping 180 features
Done saving dataset to disk
Done saving PCA and scaler to disk
```

4 Model training

[4]:
```
classifiers = {
    # 'LinearSVC': LinearSVC(dual=False),
    # 'NuSVC': NuSVC(nu=1e-07, tol=1e-09),
    # 'SGDClassifier': SGDClassifier(loss='epsilon_insensitive',␣
↪penalty='elasticnet', n_jobs=-1),
    'SVC': SVC(kernel='rbf', C=2.5, gamma=.0001105),
    # 'LinearDiscriminantAnalysis':␣
↪LinearDiscriminantAnalysis(store_covariance=True, tol=1e-06),
    'QuadraticDiscriminantAnalysis':␣
↪QuadraticDiscriminantAnalysis(store_covariance=True, tol=1e-06),
    'MLPClassifier': MLPClassifier(hidden_layer_sizes=tuple(repeat(int(pca.
↪n_components_ * 1.2), 3)), solver='lbfgs', alpha=1e-07),
    # 'DecisionTreeClassifier': DecisionTreeClassifier(),
    # 'ExtraTreeClassifier': ExtraTreeClassifier(),
    # 'KernelRidge': KernelRidge(),
    # 'GaussianProcess': GaussianProcessClassifier(1.0 * RBF(1.0)),
```

```
    'LinearRegression': LogisticRegression(dual=False, max_iter=10000, C=.01,␣
 ↪tol=1e-07)
}

models = {}
for type, classifier in classifiers.items():
    start = time.time()
    models[type] = OneVsRestClassifier(classifier, n_jobs=-1).fit(X, y)
    end = time.time()
    print(f"Done training {type} model in {'{:.2f}'.format(end - start)}s")

if save_to_disk:
    joblib.dump(models, folder + 'models')
    print("Done saving models to disk")
```

```
Done training SVC model in 1236.33s
Done training QuadraticDiscriminantAnalysis model in 45.68s
Done training MLPClassifier model in 698.50s
Done training LinearRegression model in 18.07s
Done saving models to disk
```

5 Model evaluation

[5]:
```
print('Random chance: ' + '{:.2f}%'.format(100 / len(names)))

for model_type, model in models.items():
    test2 = test.filter(regex='pixel.+').to_numpy()
    if pca_on:
        test2 = scaler.transform(test2)
        test2 = pca.transform(test2)
    prediction = model.predict(test2)

    truth = test['word'].values.tolist()
    acc_score = accuracy_score(truth, prediction)
    print(f"{model_type} classifier, accuracy: {'{:.2f}%'.format(acc_score *␣
 ↪100)}")
    print(classification_report(truth, prediction, zero_division=0))
```

```
Random chance: 2.50%
SVC classifier, accuracy: 54.72%
               precision    recall  f1-score   support

    ambulance       0.56      0.56      0.56       400
          bed       0.54      0.36      0.43       405
        bench       0.50      0.75      0.60       388
       bowtie       0.55      0.79      0.65       426
        bread       0.22      0.13      0.17       397
```

castle	0.49	0.35	0.41	377
cell phone	0.59	0.51	0.55	430
chair	0.56	0.78	0.65	408
church	0.50	0.44	0.47	372
coffee cup	0.55	0.65	0.60	405
crown	0.55	0.51	0.52	376
cruise ship	0.51	0.34	0.41	425
cup	0.44	0.27	0.34	400
dishwasher	0.48	0.65	0.55	387
dresser	0.58	0.45	0.51	400
eye	0.59	0.64	0.61	422
face	0.54	0.71	0.61	414
fan	0.48	0.64	0.55	365
fire hydrant	0.50	0.27	0.35	426
fish	0.61	0.55	0.57	404
hammer	0.60	0.67	0.63	427
hat	0.58	0.52	0.55	409
helicopter	0.61	0.66	0.63	400
ice cream	0.55	0.75	0.64	388
lantern	0.58	0.38	0.45	413
passport	0.51	0.40	0.45	398
pickup truck	0.54	0.50	0.51	404
pillow	0.62	0.83	0.71	409
power outlet	0.52	0.49	0.50	382
sailboat	0.53	0.69	0.60	383
sandwich	0.48	0.58	0.52	373
snowman	0.60	0.63	0.61	389
star	0.54	0.59	0.56	401
strawberry	0.60	0.54	0.57	372
suitcase	0.62	0.77	0.69	414
table	0.55	0.69	0.61	367
telephone	0.45	0.24	0.31	410
traffic light	0.59	0.42	0.49	408
watermelon	0.49	0.33	0.39	409
wine glass	0.70	0.89	0.78	417
accuracy			0.55	16000
macro avg	0.54	0.55	0.53	16000
weighted avg	0.54	0.55	0.53	16000

QuadraticDiscriminantAnalysis classifier, accuracy: 51.11%

	precision	recall	f1-score	support
ambulance	0.48	0.44	0.46	400
bed	0.51	0.26	0.35	405
bench	0.78	0.62	0.69	388
bowtie	0.67	0.75	0.71	426
bread	0.27	0.09	0.14	397

castle	0.38	0.40	0.39	377
cell phone	0.36	0.55	0.43	430
chair	0.70	0.75	0.72	408
church	0.40	0.51	0.45	372
coffee cup	0.57	0.44	0.50	405
crown	0.69	0.53	0.60	376
cruise ship	0.57	0.29	0.39	425
cup	0.38	0.34	0.36	400
dishwasher	0.50	0.43	0.46	387
dresser	0.46	0.59	0.52	400
eye	0.70	0.53	0.60	422
face	0.68	0.66	0.67	414
fan	0.62	0.49	0.54	365
fire hydrant	0.60	0.34	0.43	426
fish	0.20	0.68	0.31	404
hammer	0.80	0.50	0.61	427
hat	0.57	0.53	0.55	409
helicopter	0.67	0.42	0.52	400
ice cream	0.42	0.87	0.57	388
lantern	0.49	0.41	0.45	413
passport	0.40	0.29	0.34	398
pickup truck	0.22	0.64	0.33	404
pillow	0.85	0.73	0.79	409
power outlet	0.55	0.41	0.47	382
sailboat	0.65	0.64	0.64	383
sandwich	0.57	0.38	0.45	373
snowman	0.66	0.66	0.66	389
star	0.79	0.64	0.71	401
strawberry	0.46	0.44	0.45	372
suitcase	0.89	0.66	0.76	414
table	0.74	0.59	0.66	367
telephone	0.30	0.26	0.28	410
traffic light	0.49	0.42	0.45	408
watermelon	0.51	0.45	0.48	409
wine glass	0.88	0.79	0.84	417
accuracy			0.51	16000
macro avg	0.56	0.51	0.52	16000
weighted avg	0.56	0.51	0.52	16000

MLPClassifier classifier, accuracy: 59.30%

	precision	recall	f1-score	support
ambulance	0.53	0.54	0.53	400
bed	0.53	0.46	0.49	405
bench	0.69	0.69	0.69	388
bowtie	0.84	0.78	0.81	426
bread	0.28	0.20	0.24	397

castle	0.38	0.41	0.40	377
cell phone	0.58	0.52	0.55	430
chair	0.73	0.79	0.76	408
church	0.44	0.48	0.46	372
coffee cup	0.52	0.61	0.57	405
crown	0.47	0.55	0.51	376
cruise ship	0.48	0.42	0.45	425
cup	0.41	0.40	0.40	400
dishwasher	0.58	0.60	0.59	387
dresser	0.51	0.53	0.52	400
eye	0.74	0.71	0.73	422
face	0.59	0.77	0.67	414
fan	0.64	0.65	0.65	365
fire hydrant	0.47	0.39	0.43	426
fish	0.62	0.59	0.60	404
hammer	0.70	0.71	0.71	427
hat	0.60	0.56	0.58	409
helicopter	0.63	0.64	0.64	400
ice cream	0.78	0.86	0.82	388
lantern	0.55	0.44	0.49	413
passport	0.48	0.46	0.47	398
pickup truck	0.56	0.54	0.55	404
pillow	0.79	0.79	0.79	409
power outlet	0.54	0.55	0.55	382
sailboat	0.66	0.66	0.66	383
sandwich	0.57	0.61	0.59	373
snowman	0.70	0.73	0.71	389
star	0.61	0.57	0.59	401
strawberry	0.59	0.61	0.60	372
suitcase	0.71	0.80	0.75	414
table	0.62	0.69	0.66	367
telephone	0.42	0.40	0.41	410
traffic light	0.64	0.60	0.62	408
watermelon	0.55	0.48	0.51	409
wine glass	0.82	0.89	0.86	417
accuracy			0.59	16000
macro avg	0.59	0.59	0.59	16000
weighted avg	0.59	0.59	0.59	16000

LinearRegression classifier, accuracy: 42.39%

	precision	recall	f1-score	support
ambulance	0.39	0.44	0.41	400
bed	0.28	0.17	0.21	405
bench	0.49	0.67	0.57	388
bowtie	0.53	0.74	0.61	426
bread	0.12	0.03	0.05	397

```
        castle       0.25      0.23      0.24       377
    cell phone       0.40      0.47      0.44       430
         chair       0.50      0.67      0.57       408
        church       0.31      0.27      0.29       372
    coffee cup       0.41      0.42      0.42       405
         crown       0.32      0.30      0.31       376
   cruise ship       0.29      0.23      0.26       425
           cup       0.31      0.25      0.28       400
    dishwasher       0.51      0.56      0.54       387
       dresser       0.37      0.26      0.31       400
           eye       0.46      0.55      0.50       422
          face       0.52      0.58      0.55       414
           fan       0.48      0.50      0.49       365
  fire hydrant       0.34      0.18      0.24       426
          fish       0.34      0.46      0.39       404
        hammer       0.45      0.41      0.43       427
           hat       0.38      0.43      0.40       409
    helicopter       0.29      0.28      0.29       400
     ice cream       0.55      0.72      0.63       388
       lantern       0.38      0.33      0.35       413
      passport       0.32      0.22      0.26       398
  pickup truck       0.33      0.43      0.38       404
        pillow       0.67      0.80      0.73       409
  power outlet       0.33      0.31      0.32       382
      sailboat       0.46      0.57      0.51       383
      sandwich       0.38      0.40      0.39       373
       snowman       0.47      0.58      0.52       389
          star       0.36      0.42      0.39       401
    strawberry       0.35      0.35      0.35       372
      suitcase       0.66      0.70      0.68       414
         table       0.40      0.53      0.46       367
     telephone       0.20      0.08      0.11       410
 traffic light       0.42      0.30      0.35       408
    watermelon       0.29      0.24      0.26       409
    wine glass       0.76      0.84      0.80       417

      accuracy                           0.42     16000
     macro avg       0.40      0.42      0.41     16000
  weighted avg       0.40      0.42      0.41     16000
```

[ ]:
```python
cls_type, model = random.choice(list(models.items()))
# cls_type = 'MLPClassifier'
# model = models[list(models.keys())[0]]

sample = test.sample(1)
sample_predict = sample.filter(regex='pixel.+').to_numpy()
```

```python
if pca_on:
    sample_predict = scaler.transform(sample_predict)
    sample_predict = pca.transform(sample_predict)

prediction = model.predict(sample_predict)
display(IPImage(render_single(sample['drawing'].iloc[0])))
print(f"Using {cls_type} classifier")
print(f"{prediction[0]}(predicted) == {sample['word'].iloc[0]}(actual) ? {sample['word'].iloc[0] == prediction[0]}")
```