# 05-predict-country

December 17, 2021

[4]:
```
%load_ext autoreload
%autoreload 2
import time
from utils import generate_pixel_columns, extract_best_entries,␣
↪extract_first_entries, equalize_by
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[7]:
```
names = ['power outlet']
    # ['ambulance','bed','bench','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']
image_gen_params = {
    'magnification': 4,
    'resolution': 32,
    'invert_color': True,
    'stroke_width_scale': 2
}

files = list(map(lambda n: f"./dataset/{n}.ndjson", names))

df = extract_best_entries(files, recognized=True, skip_first=200)
df = equalize_by(df, 'countrycode', num_entries=500)
print(f"Retained {len(df['countrycode'].value_counts())} countries")
df = generate_pixel_columns(df, **image_gen_params)
```

```
3000
Retained 1 countries
```

[33]:
```
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.')
```

```
Train: 40500 entries, test: 4500 entries.
```

[34]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neural_network import MLPClassifier
from itertools import repeat
import time

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

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')

classifier = MLPClassifier(hidden_layer_sizes=tuple(repeat(int(pca.
 ↪n_components_ * 1.2), 3)), solver='lbfgs', alpha=1e-07)
start = time.time()
model = OneVsRestClassifier(classifier, n_jobs=-1).fit(X, y)
end = time.time()
print(f"Done training model in {'{:.2f}'.format(end - start)}s")
```

```
Done generating features and target
PCA & standardization done. Keeping 585 features
Done training model in 6178.25s
```

[35]:
```
test2 = test.filter(regex='pixel.+').to_numpy()
test2 = scaler.transform(test2)
test2 = pca.transform(test2)
prediction = model.predict(test2)

countries = list(test['countrycode'].value_counts().keys())
counts = {}
for idx in range(len(test)):
    country = test['countrycode'].iloc[idx]
    entry_score = counts.get(country, (0, 0))
```

```
    entry_score = (entry_score[0] + 1 if prediction[idx] == country else␣
 ↪entry_score[0], entry_score[1] + 1)
    counts[country] = entry_score
scores = {}
for country in countries:
    scores[country] = (counts[country][0] / counts[country][1]) * 100

scores = [(k, v) for k, v in scores.items()]
scores.sort(key=lambda e : e[1], reverse=True)
threshold = 100 / len(countries)

print(f'Scores greater than {"{:.2f}".format(threshold)}% (random chance):')
for entry in scores:
    if entry[1] > threshold:
        print(f'  {entry[0]}: {"{:.2f}".format(entry[1])}%')

from sklearn.metrics import accuracy_score
acc_score = accuracy_score(test['countrycode'].values.tolist(), prediction)
print(f"Overall accuracy: {'{:.2f}'.format(acc_score)}%")
```

```
Scores greater than 2.22% (random chance):
  TW: 14.04%
  JP: 8.62%
  KR: 7.61%
  AT: 7.37%
  CA: 6.60%
  TH: 6.06%
  PL: 5.75%
  MX: 4.72%
  AU: 4.49%
  MY: 4.35%
  ID: 3.64%
  NO: 3.26%
  SE: 3.06%
  FR: 2.97%
  SA: 2.91%
  RU: 2.86%
  BR: 2.80%
  AE: 2.75%
  NZ: 2.75%
  PH: 2.68%
  FI: 2.50%
  US: 2.27%
  RS: 2.27%
Overall accuracy: 0.03%
```