Modele3_debug

November 18, 2025

[1]:
```python
import sys
import numpy as np
import pickle
np.random.seed(0)
from scripts import (
    train_test_split,
    StandardScaler,
    accuracy,
    confusion_matrix,
    recall_per_class,
    balanced_accuracy


)

import pandas as pd
```

[2]:
```python
def rbf_kernel(x, X, sigma=1.0):
    """Calcule le noyau RBF entre x et X."""
    return np.exp(-np.sum((X - x)**2, axis=1) / (2 * sigma**2))


class KernelPerceptron:
    def __init__(self, kernel_fn, n_classes, sigma=1.0):
        self.kernel_fn = kernel_fn
        self.n_classes = n_classes
        self.sigma = sigma


    def fit(self, X, y, max_epochs=10):
        """Entraînement multiclasse One-vs-Rest."""
        N = X.shape[0]
        self.X = X
        self.classes = np.unique(y)
        self.train_y = y
        self.alpha = np.zeros((self.n_classes, N))

        # Precompute Gram matrix
```

```
        self.K = np.array([self.kernel_fn(X[i], X, self.sigma) for i in␣
↪range(N)])

        for c in self.classes:
            print(f"Training classifier for class {c}...")
            y_bin = np.where(y == c, 1, -1)
            count, i, n_iter = 0, 0, 0

            while count < N and n_iter < max_epochs * N:
                if np.sign(np.sum(self.alpha[c] * y_bin * self.K[i])) !=␣
↪y_bin[i]:
                    self.alpha[c][i] += 1
                    count = 0
                else:
                    count += 1
                i = (i + 1) % N
                n_iter += 1
        print("Training done.")

    def predict(self, X_test):
        """Retourne la classe prédite pour chaque x_test."""
        scores = np.zeros((len(X_test), self.n_classes))
        for i, x in enumerate(X_test):
            k = self.kernel_fn(x, self.X, self.sigma)
            for c in self.classes:
                y_bin = np.where(self.train_y == c, 1, -1)
                scores[i, c] = np.sum(self.alpha[c] * y_bin * k)
        return np.argmax(scores, axis=1)
```

[3]:
```
def rbf_kernel_svm(X,Y, sigma):
    dists = np.sum(X**2, axis=1)[:,None] + np.sum(Y**2, axis=1)[None, :] - 2 *␣
↪X @ Y.T
    return np.exp(-dists / (2*sigma**2))

class SVM:
    def __init__(self, sigma=1.0, max_iter=5, sample_weights=None):
        self.sigma = sigma
        self.max_iter = max_iter
        self.sample_weights = sample_weights

    def fit(self, X, y):
        self.X = X
        self.y = y
        self.classes = np.unique(y)
        self.K = rbf_kernel_svm(X,X,self.sigma)

        N = X.shape[0]
```

```python
        self.alpha = {}

        for c in self.classes:
            print(f"train class {c} vs rest")
            y_bin = np.where(y == c, 1, -1)
            alpha_c = np.zeros(N)

            for it in range(self.max_iter):
                for i in range(N):
                    f = np.sum(alpha_c * y_bin * self.K[:, i])
                    if y_bin[i] * f <= 0:
                        alpha_c[i] += self.sample_weights[i]

            self.alpha[c] = alpha_c

    def decision_function(self, X_test):
        K_test = rbf_kernel_svm(X_test, self.X, self.sigma)

        scores = []
        for c in self.classes:
            y_bin = np.where(self.y == c,1,-1)
            s = K_test @ (self.alpha[c]*y_bin)
            scores.append(s)

        return np.vstack(scores).T

    def predict(self, X_test):
        scores = self.decision_function(X_test)
        return self.classes[np.argmax(scores, axis=1)]
```

[4]:
```python
# --- Load training data ---
path_to_data = 'ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl'
with open(path_to_data, "rb") as f:
    train_data = pickle.load(f)

X_imgs = train_data["images"].astype(np.float32)

y = train_data["labels"].reshape(-1)
X_imgs = (X_imgs - X_imgs.min()) / (X_imgs.max() - X_imgs.min() + 1e-6)
X_imgs = (X_imgs - X_imgs.mean()) / (X_imgs.std() + 1e-6)
```

[5]:
```python
def radial_profile(img):
    """Calcule le profil radial moyen d'une image."""
    h, w = img.shape
    y, x = np.ogrid[:h, :w]
    r = np.sqrt((x - w//2)**2 + (y - h//2)**2).astype(int)
```

```python
    return np.bincount(r.ravel(), img.ravel()) / np.bincount(r.ravel())
```

[6]:
```python
def extract_simple_stats(img):
    gray = img.mean(axis=2)
    return np.array([gray.mean(), gray.std(), gray.min(), gray.max()], dtype=np.
↪float32)
```

[7]:
```python
def simple_augment(images):
    flips = images[:, :, ::-1, :]   # inversion horizontale
    noise = images + 0.01*np.random.randn(*images.shape)
    return np.concatenate([images, flips,
                            noise
                            ],
                              axis=0)
```

[8]:
```python
def fft_features(images):
    """Extrait les caractéristiques FFT d'images."""
    gray = images.mean(axis=3)
    F = np.fft.fft2(gray, axes=(1, 2))
    return np.abs(np.fft.fftshift(F, axes=(1, 2)))
```

[9]:
```python
X_imgs = simple_augment(X_imgs)
y = np.concatenate([y, y, y])

fft_mag = fft_features(X_imgs)
X_fft = np.array([radial_profile(img) for img in fft_mag], dtype=np.float32)

X_stats = np.array([extract_simple_stats(img) for img in X_imgs], dtype=np.
↪float32)
X = np.hstack([X_fft, X_stats])

scaler = StandardScaler()
X = scaler.fit_transform(X)
```

[10]:
```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0
)
```

[11]:
```python
"""
model = SVM(sigma=5, max_iter=5, sample_weights=sample_weights)
model.fit(X_train, y_train)
"""
```

[11]:
```
'\nmodel = SVM(sigma=5, max_iter=5,
sample_weights=sample_weights)\nmodel.fit(X_train, y_train)\n'
```

[12]:
```
model = KernelPerceptron(kernel_fn=rbf_kernel, n_classes=5, sigma=5)
model.fit(X_train, y_train, max_epochs=5)
```

```
Training classifier for class 0…
Training classifier for class 1…
Training classifier for class 2…
Training classifier for class 3…
Training classifier for class 4…
Training done.
```

[13]:
```
"""
model = SoftmaxClassifier(input_dim=X.shape[1], num_classes=num_classes, reg=0.
↪05, seed=0)
model.fit(X_train, y_train, lr=2, n_steps=5000, sample_weights=sample_weights)
"""
```

[13]:
```
'\nmodel = SoftmaxClassifier(input_dim=X.shape[1], num_classes=num_classes,
reg=0.05, seed=0)\nmodel.fit(X_train, y_train, lr=2, n_steps=5000,
sample_weights=sample_weights)\n'
```

Understanding Deep Learning Requires Rethinking Generalization (Zhang et al., ICLR 2017) → importance de la fréquence dans image classification. Fourier Transform in Image Processing — Gonzalez & Woods (exemples de séparation de classes via magnitude spectrale) FOURIER CNNs (Rippel et al., 2015) → montre qu'une représentation FFT est plus stable et expressive que pixels bruts. Spectral Representations for Image Classification (several IEEE papers) → radial power spectra suffisent à classifier des textures complexes. Why Do Deep Neural Networks Learn High-Frequency Patterns? (Xu et al., NeurIPS 2019)

[14]:
```
y_pred = model.predict(X_test)
acc = (y_pred == y_test).mean()
print("Test accuracy =", acc)

cm = confusion_matrix(y_test, y_pred)
bal_acc = balanced_accuracy(y_test, y_pred)
rec = recall_per_class(cm)

print("Balanced acc :", bal_acc)
print("Recall par classe :", rec)
print("Recall moyen :", rec.mean())
print(cm)
```

```
Test accuracy = 0.3888888888888889
Balanced acc : 0.3408322856598927
Recall par classe : [0.49822064 0.05970149 0.64122137 0.01612903 0.48888889]
Recall moyen : 0.3408322856598927
[[140   1 113   1  26]
 [ 17   4  36   0  10]
 [ 19   3  84   1  24]
 [ 19   0  78   2  25]
```

```
[  4   0  19   0  22]]
```

[15]:
```
pickle.dump((model, scaler), open("model_test1.pkl", "wb"))
```

[ ]:
```
# --------------------------
# 1. Charger le modèle entraîné
# --------------------------
model, scaler = pickle.load(open("model_test1.pkl", "rb"))

# --------------------------
# 2. Charger le test_data.pkl
# --------------------------
with open("ift-3395-6390-kaggle-2-competition-fall-2025/test_data.pkl", "rb")␣
↪as f:
    test_data = pickle.load(f)

X_test_imgs = test_data["images"]


# Apply to test set
#X_test_feats = np.array([extract_features(img) for img in X_test_imgs],␣
↪dtype=np.float32)

# --------------------------
# 4. Normaliser avec les stats du train
# --------------------------
fft_mag_test = fft_features(X_test_imgs)
X_fft_test = np.array([radial_profile(img) for img in fft_mag_test], dtype=np.
↪float32)

X_stats_test = np.array([extract_simple_stats(img) for img in X_test_imgs],␣
↪dtype=np.float32)

X_test_feats = np.hstack([X_fft_test, X_stats_test])

X_test_norm = scaler.transform(X_test_feats)

# --------------------------
# 5. Prédire
# --------------------------

y_pred = model.predict(X_test_norm).astype(int)

# --------------------------
# 6. Générer le CSV Kaggle
# --------------------------
df = pd.DataFrame({
```

```python
    "ID": np.arange(1, len(y_pred)+1),
    "Label": y_pred
})

df.to_csv("ift3395_YAPS_MCS_V12.csv", index=False)

print("Fichier 'submission.csv' généré !")

print(df.head())

df1 = pd.read_csv("ift3395_YAPS_MCS_V12.csv.csv")
df2 = pd.read_csv("ift3395_YAPS_MCS_V10.csv")

comparison = df1.compare(df2)
print(comparison)
print("Nombre de différences :", len(comparison))
```

```
Fichier 'submission.csv' généré !
   ID  Label
0   1      0
1   2      0
2   3      0
3   4      0
4   5      0
Empty DataFrame
Columns: []
Index: []
Nombre de différences : 0
```