CB-M1-ARCH2506-0327

Task 1

MNIST Project:https://github.com/Taqadus842/AI-ML/tree/main/mnist%20digit%20recognition

---

Chapter 3 Solution:

1. MNIST Classifier with >97% Accuracy using KNeighborsClassifier

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Load MNIST
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist["data"], mnist["target"].astype(int)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10000,
random_state=42)

# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Grid search for hyperparameter tuning
param_grid = {
    'n_neighbors': [3, 4, 5],
    'weights': ['uniform', 'distance']
}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=3, verbose=1)
grid_search.fit(X_train, y_train)

# Evaluate
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
```

Expected Accuracy: 97–98% depending on hyperparameters and preprocessing.

---

2. Shift MNIST Images to Augment Data

Shift Function:

```
from scipy.ndimage import shift

def shift_image(image, dx, dy):
    return shift(image.reshape(28, 28), [dy, dx], cval=0).reshape(784)
```

Data Augmentation:

```
import numpy as np

X_augmented = []
y_augmented = []

for img, label in zip(X_train, y_train):
    X_augmented.append(img)  # original
    y_augmented.append(label)
    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        shifted = shift_image(img, dx, dy)
        X_augmented.append(shifted)
        y_augmented.append(label)

X_train_aug = np.array(X_augmented)
y_train_aug = np.array(y_augmented)
```

Retrain Model on Augmented Set:

```
X_train_aug = scaler.fit_transform(X_train_aug)
knn = KNeighborsClassifier(n_neighbors=3, weights='distance')
knn.fit(X_train_aug, y_train_aug)
y_pred = knn.predict(X_test)
print("Accuracy after augmentation:", accuracy_score(y_test, y_pred))
```

Expected Accuracy Boost: 98%+ due to better generalization.

---

3. Titanic Dataset Classification (Kaggle)

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load
df = pd.read_csv("titanic.csv")

# Preprocess
df["Sex"] = df["Sex"].map({"male": 0, "female": 1})
df["Embarked"].fillna("S", inplace=True)
df["Embarked"] = df["Embarked"].map({"S": 0, "C": 1, "Q": 2})
df["Age"].fillna(df["Age"].median(), inplace=True)
df["Fare"].fillna(df["Fare"].median(), inplace=True)

features = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]
X = df[features]
y = df["Survived"]
```

```
# Train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Titanic Accuracy:", accuracy_score(y_test, y_pred))
```

---

4. Spam Classifier from Apache SpamAssassin

Preprocessing Pipeline:

```
import os
import email
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

def load_emails(path):
    emails = []
    for filename in os.listdir(path):
        with open(os.path.join(path, filename), 'rb') as f:
            msg = email.message_from_binary_file(f)
            payload = msg.get_payload()
            if isinstance(payload, list):
                body = ''.join([part.get_payload() for part in payload])
            else:
                body = payload
            emails.append(body)
    return emails

spam = load_emails("path/to/spam")
ham = load_emails("path/to/ham")
X = spam + ham
y = [1]*len(spam) + [0]*len(ham)

# Clean and vectorize
def preprocess(text):
    text = text.lower()
    text = re.sub(r'\W+', ' ', text)
    return text

X_cleaned = [preprocess(text) for text in X]
vectorizer = CountVectorizer()
X_vect = vectorizer.fit_transform(X_cleaned)

# Train/Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_vect, y, test_size=0.2,
random_state=42)

model = MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

Aim for **high precision & recall**. You can add stemming, URL/number handling, etc., using `nltk`, `re`, or `scikit-learn` custom transformers.
