```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 17 10:21:48 2022

@credits: https://github.com/ageron/handson-ml/blob/master/05_support_vector_machines.ipynb
"""
##train an SVM classifier on the MNIST dataset. Since SVM classifiers are binary classifiers, you will need to use one-versus-all to classify
all 10 digits. You may want to tune the hyperparameters using small validation sets to speed up the process. What accuracy can you reach?
#First, let's load the dataset and split it into a training set and a test set. We could use train_test_split() but people usually just take the
first 60,000 instances for the training set, and the last 10,000 instances for the test set (this makes it possible to compare your model's
performance with others):

# To support both python 2 and python 3

# Common imports
import numpy as np

# to make this notebook's output stable across runs
np.random.seed(42)
from sklearn.svm import SVC, LinearSVC
from sklearn.preprocessing import StandardScaler

try:
    from sklearn.datasets import fetch_openml
    mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)
except ImportError:
    from sklearn.datasets import fetch_mldata
    mnist = fetch_mldata('MNIST original')

X = mnist["data"]
y = mnist["target"]

X_train = X[:60000]
y_train = y[:60000]
X_test = X[60000:]
y_test = y[60000:]
#Many training algorithms are sensitive to the order of the training instances, so it's generally good practice to shuffle them first:


np.random.seed(42)
rnd_idx = np.random.permutation(60000)
X_train = X_train[rnd_idx]
y_train = y_train[rnd_idx]

#Let's start simple, with a linear SVM classifier. It will automatically use the One-vs-All (also called One-vs-the-Rest, OvR) strategy, so
there's nothing special we need to do. Easy!
```

```
lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train, y_train)

#Let's make predictions on the training set and measure the accuracy (we don't want to measure it on the test set yet, since we have not
selected and trained the final model yet):

from sklearn.metrics import accuracy_score

y_pred = lin_clf.predict(X_train)
accuracy_score(y_train, y_pred)

#86% accuracy on MNIST is a really bad performance. This linear model is certainly too simple for MNIST, but perhaps we just needed to scale the
data first:

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float32))
X_test_scaled = scaler.transform(X_test.astype(np.float32))

lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train_scaled, y_train)

y_pred = lin_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)

# 93% accuracy, That's much better (we cut the error rate in two), but still not great at all for MNIST. If we want to use an SVM, we will have
to use a kernel. Let's try an SVC with an RBF kernel (the default).

#Warning: if you are using Scikit-Learn ≤ 0.19, the SVC class will use the One-vs-One (OvO) strategy by default, so you must explicitly set
decision_function_shape="ovr" if you want to use the OvR strategy instead (OvR is the default since 0.19).

svm_clf = SVC(decision_function_shape="ovr", gamma="auto")
svm_clf.fit(X_train_scaled[:10000], y_train[:10000])

y_pred = svm_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)

# check if you have got better accuracy now

#we get better performance even though we trained the model on 6 times less data. Let's tune the hyperparameters by doing a randomized search
with cross validation. We will do this on a small dataset just to speed up the process:


from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform

param_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1, 10)}
```

```
rnd_search_cv = RandomizedSearchCV(svm_clf, param_distributions, n_iter=10, verbose=2, cv=3)
rnd_search_cv.fit(X_train_scaled[:1000], y_train[:1000])

rnd_search_cv.best_estimator_
rnd_search_cv.best_score_

#This looks pretty low but remember we only trained the model on 1,000 instances. Let's retrain the best estimator on the whole training set
(run this at night, it will take hours):

rnd_search_cv.best_estimator_.fit(X_train_scaled, y_train)

y_pred = rnd_search_cv.best_estimator_.predict(X_train_scaled)
accuracy_score(y_train, y_pred)

#99% accuracy this looks good! Let's select this model. Now we can test it on the test set:

y_pred = rnd_search_cv.best_estimator_.predict(X_test_scaled)
accuracy_score(y_test, y_pred)

#Not too bad, but apparently the model is overfitting slightly. It's tempting to tweak the hyperparameters a bit more (e.g. decreasing C and/or
gamma), but we would run the risk of overfitting the test set. Other people have found that the hyperparameters C=5 and gamma=0.005 yield even
better performance (over 98% accuracy). By running the randomized search for longer and on a larger part of the training set, you may be able to
find this as well.
```