AAI_501_assignment_6.1_ajmal-jalal.pdf

December 2, 2024

1 Hand-Written Digits Recognition

As you have learned and practiced in the module 4, Scikit-learn (package name: `sklearn`) is a Python-based toolkit for a wide variety of machine learning methods. It supports both supervised learning methods such as classification and regression and unsupervised methods such as clustering and dimensionality reduction. In addition, it provides much of the software infrastructure to enable the construction of machine learning pipelines, supporting activities such as train-test splitting, cross validation, and hyperparameter optimization.

In addition, the scikit-learn developer community has prepared rich documentation that not only describes the particular functions supported by the package, but also theoretical background on different machine learning algorithms and their relationship to one another.

In this assignment, you will continue to explore scikit-learn functionalities to tackle a popular computer vision challenge to recognize hand-written digits.

1.0.1 Step 1.

Let's begin by importing `sklearn`. Execute the command `help(sklearn)` to get a brief introduction to the package. You should see in the package description that sklearn in intended to work in the "tightly-knit world of scientific Python packages (numpy, scipy, matplotlib)." This is part of what makes a successful ecosystem: higher-level packages implementating more specific functionality such as machine learning are able to rest on top of the infrastructure provided by other more general packages in the ecosystem.

You should also notice that `sklearn` is a Python module supporting "classical machine learning algorithms." Machine-learning algorithms have been developed and used for decades, although much of the recent excitement around the field revolves around newer "deep-learning" algorithms that use neural networks for their implementation. While `sklearn` does provide some support for machine learning using neural networks, that is not its primary focus, hence the emphasis on "classical" algorithms. There is no single algorithm or approach that works best for every problem, and one of the strengths of `sklearn` is that it supports the use and comparison of many different algorithms within one consistent and integrated framework.

[1]:
```
import sklearn
help(sklearn)
```

```
Help on package sklearn:

NAME
```

```
    sklearn - Configure global settings and get information about the working
environment.

PACKAGE CONTENTS
    __check_build (package)
    _build_utils (package)
    _built_with_meson
    _config
    _distributor_init
    _isotonic
    _loss (package)
    _min_dependencies
    base
    calibration
    cluster (package)
    compose (package)
    conftest
    covariance (package)
    cross_decomposition (package)
    datasets (package)
    decomposition (package)
    discriminant_analysis
    dummy
    ensemble (package)
    exceptions
    experimental (package)
    externals (package)
    feature_extraction (package)
    feature_selection (package)
    gaussian_process (package)
    impute (package)
    inspection (package)
    isotonic
    kernel_approximation
    kernel_ridge
    linear_model (package)
    manifold (package)
    metrics (package)
    mixture (package)
    model_selection (package)
    multiclass
    multioutput
    naive_bayes
    neighbors (package)
    neural_network (package)
    pipeline
    preprocessing (package)
    random_projection
```

```
    semi_supervised (package)
    svm (package)
    tests (package)
    tree (package)
    utils (package)

FUNCTIONS
    clone(estimator, *, safe=True)
        Construct a new unfitted estimator with the same parameters.

        Clone does a deep copy of the model in an estimator
        without actually copying attached data. It returns a new estimator
        with the same parameters that has not been fitted on any data.

        .. versionchanged:: 1.3
            Delegates to `estimator.__sklearn_clone__` if the method exists.

        Parameters
        ----------
        estimator : {list, tuple, set} of estimator instance or a single
estimator instance
            The estimator or group of estimators to be cloned.
        safe : bool, default=True
            If safe is False, clone will fall back to a deep copy on objects
            that are not estimators. Ignored if `estimator.__sklearn_clone__`
            exists.

        Returns
        -------
        estimator : object
            The deep copy of the input, an estimator if input is an estimator.

        Notes
        -----
        If the estimator's `random_state` parameter is an integer (or if the
        estimator doesn't have a `random_state` parameter), an *exact clone* is
        returned: the clone and the original estimator will give the exact same
        results. Otherwise, *statistical clone* is returned: the clone might
        return different results from the original estimator. More details can
be
        found in :ref:`randomness`.

        Examples
        --------
        >>> from sklearn.base import clone
        >>> from sklearn.linear_model import LogisticRegression
        >>> X = [[-1, 0], [0, 1], [0, -1], [1, 0]]
        >>> y = [0, 0, 1, 1]
```

```
        >>> classifier = LogisticRegression().fit(X, y)
        >>> cloned_classifier = clone(classifier)
        >>> hasattr(classifier, "classes_")
        True
        >>> hasattr(cloned_classifier, "classes_")
        False
        >>> classifier is cloned_classifier
        False

    config_context(*, assume_finite=None, working_memory=None,
print_changed_only=None, display=None, pairwise_dist_chunk_size=None,
enable_cython_pairwise_dist=None, array_api_dispatch=None,
transform_output=None, enable_metadata_routing=None,
skip_parameter_validation=None)
        Context manager for global scikit-learn configuration.

        Parameters
        ----------
        assume_finite : bool, default=None
            If True, validation for finiteness will be skipped,
            saving time, but leading to potential crashes. If
            False, validation for finiteness will be performed,
            avoiding error. If None, the existing value won't change.
            The default value is False.

        working_memory : int, default=None
            If set, scikit-learn will attempt to limit the size of temporary
arrays
            to this number of MiB (per job when parallelised), often saving both
            computation time and memory on expensive operations that can be
            performed in chunks. If None, the existing value won't change.
            The default value is 1024.

        print_changed_only : bool, default=None
            If True, only the parameters that were set to non-default
            values will be printed when printing an estimator. For example,
            ``print(SVC())`` while True will only print 'SVC()', but would print
            'SVC(C=1.0, cache_size=200, …)' with all the non-changed
parameters
            when False. If None, the existing value won't change.
            The default value is True.

            .. versionchanged:: 0.23
               Default changed from False to True.

        display : {'text', 'diagram'}, default=None
            If 'diagram', estimators will be displayed as a diagram in a Jupyter
            lab or notebook context. If 'text', estimators will be displayed as
```

```
            text. If None, the existing value won't change.
            The default value is 'diagram'.

            .. versionadded:: 0.23

        pairwise_dist_chunk_size : int, default=None
            The number of row vectors per chunk for the accelerated pairwise-
            distances reduction backend. Default is 256 (suitable for most of
            modern laptops' caches and architectures).

            Intended for easier benchmarking and testing of scikit-learn
internals.
            End users are not expected to benefit from customizing this
configuration
            setting.

            .. versionadded:: 1.1

        enable_cython_pairwise_dist : bool, default=None
            Use the accelerated pairwise-distances reduction backend when
            possible. Global default: True.

            Intended for easier benchmarking and testing of scikit-learn
internals.
            End users are not expected to benefit from customizing this
configuration
            setting.

            .. versionadded:: 1.1

        array_api_dispatch : bool, default=None
            Use Array API dispatching when inputs follow the Array API standard.
            Default is False.

            See the :ref:`User Guide <array_api>` for more details.

            .. versionadded:: 1.2

        transform_output : str, default=None
            Configure output of `transform` and `fit_transform`.

            See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
            for an example on how to use the API.

            - `"default"`: Default output format of a transformer
            - `"pandas"`: DataFrame output
            - `"polars"`: Polars output
            - `None`: Transform configuration is unchanged
```

```
            .. versionadded:: 1.2
            .. versionadded:: 1.4
                `"polars"` option was added.

        enable_metadata_routing : bool, default=None
            Enable metadata routing. By default this feature is disabled.

            Refer to :ref:`metadata routing user guide <metadata_routing>` for
more
            details.

            - `True`: Metadata routing is enabled
            - `False`: Metadata routing is disabled, use the old syntax.
            - `None`: Configuration is unchanged

            .. versionadded:: 1.3

        skip_parameter_validation : bool, default=None
            If `True`, disable the validation of the hyper-parameters' types and
values in
            the fit method of estimators and for arguments passed to public
helper
            functions. It can save time in some situations but can lead to low
level
            crashes and exceptions with confusing error messages.

            Note that for data parameters, such as `X` and `y`, only type
validation is
            skipped but validation with `check_array` will continue to run.

            .. versionadded:: 1.3

        Yields
        ------
        None.

        See Also
        --------
        set_config : Set global scikit-learn configuration.
        get_config : Retrieve current values of the global configuration.

        Notes
        -----
        All settings, not just those presently modified, will be returned to
        their previous values when the context manager is exited.

        Examples
```

```
        --------
        >>> import sklearn
        >>> from sklearn.utils.validation import assert_all_finite
        >>> with sklearn.config_context(assume_finite=True):
        …     assert_all_finite([float('nan')])
        >>> with sklearn.config_context(assume_finite=True):
        …     with sklearn.config_context(assume_finite=False):
        …         assert_all_finite([float('nan')])
        Traceback (most recent call last):
        …
        ValueError: Input contains NaN…

    get_config()
        Retrieve current values for configuration set by :func:`set_config`.

        Returns
        -------
        config : dict
            Keys are parameter names that can be passed to :func:`set_config`.

        See Also
        --------
        config_context : Context manager for global scikit-learn configuration.
        set_config : Set global scikit-learn configuration.

        Examples
        --------
        >>> import sklearn
        >>> config = sklearn.get_config()
        >>> config.keys()
        dict_keys([…])

    set_config(assume_finite=None, working_memory=None, print_changed_only=None,
display=None, pairwise_dist_chunk_size=None, enable_cython_pairwise_dist=None,
array_api_dispatch=None, transform_output=None, enable_metadata_routing=None,
skip_parameter_validation=None)
        Set global scikit-learn configuration.

        .. versionadded:: 0.19

        Parameters
        ----------
        assume_finite : bool, default=None
            If True, validation for finiteness will be skipped,
            saving time, but leading to potential crashes. If
            False, validation for finiteness will be performed,
            avoiding error.  Global default: False.
```

```
            .. versionadded:: 0.19

        working_memory : int, default=None
            If set, scikit-learn will attempt to limit the size of temporary
arrays
            to this number of MiB (per job when parallelised), often saving both
            computation time and memory on expensive operations that can be
            performed in chunks. Global default: 1024.

            .. versionadded:: 0.20

        print_changed_only : bool, default=None
            If True, only the parameters that were set to non-default
            values will be printed when printing an estimator. For example,
            ``print(SVC())`` while True will only print 'SVC()' while the
default
            behaviour would be to print 'SVC(C=1.0, cache_size=200, …)' with
            all the non-changed parameters.

            .. versionadded:: 0.21

        display : {'text', 'diagram'}, default=None
            If 'diagram', estimators will be displayed as a diagram in a Jupyter
            lab or notebook context. If 'text', estimators will be displayed as
            text. Default is 'diagram'.

            .. versionadded:: 0.23

        pairwise_dist_chunk_size : int, default=None
            The number of row vectors per chunk for the accelerated pairwise-
            distances reduction backend. Default is 256 (suitable for most of
            modern laptops' caches and architectures).

            Intended for easier benchmarking and testing of scikit-learn
internals.
            End users are not expected to benefit from customizing this
configuration
            setting.

            .. versionadded:: 1.1

        enable_cython_pairwise_dist : bool, default=None
            Use the accelerated pairwise-distances reduction backend when
            possible. Global default: True.

            Intended for easier benchmarking and testing of scikit-learn
internals.
            End users are not expected to benefit from customizing this
```

```
configuration
            setting.

            .. versionadded:: 1.1

        array_api_dispatch : bool, default=None
            Use Array API dispatching when inputs follow the Array API standard.
            Default is False.

            See the :ref:`User Guide <array_api>` for more details.

            .. versionadded:: 1.2

        transform_output : str, default=None
            Configure output of `transform` and `fit_transform`.

            See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
            for an example on how to use the API.

            - `"default"`: Default output format of a transformer
            - `"pandas"`: DataFrame output
            - `"polars"`: Polars output
            - `None`: Transform configuration is unchanged

            .. versionadded:: 1.2
            .. versionadded:: 1.4
                `"polars"` option was added.

        enable_metadata_routing : bool, default=None
            Enable metadata routing. By default this feature is disabled.

            Refer to :ref:`metadata routing user guide <metadata_routing>` for
more
            details.

            - `True`: Metadata routing is enabled
            - `False`: Metadata routing is disabled, use the old syntax.
            - `None`: Configuration is unchanged

            .. versionadded:: 1.3

        skip_parameter_validation : bool, default=None
            If `True`, disable the validation of the hyper-parameters' types and
values in
            the fit method of estimators and for arguments passed to public
helper
            functions. It can save time in some situations but can lead to low
level
```

```
            crashes and exceptions with confusing error messages.

            Note that for data parameters, such as `X` and `y`, only type
validation is
            skipped but validation with `check_array` will continue to run.

            .. versionadded:: 1.3

        See Also
        --------
        config_context : Context manager for global scikit-learn configuration.
        get_config : Retrieve current values of the global configuration.

        Examples
        --------
        >>> from sklearn import set_config
        >>> set_config(display='diagram')  # doctest: +SKIP

    show_versions()
        Print useful debugging information"

        .. versionadded:: 0.20

        Examples
        --------
        >>> from sklearn import show_versions
        >>> show_versions()  # doctest: +SKIP

DATA
    __SKLEARN_SETUP__ = False
    __all__ = ['calibration', 'cluster', 'covariance', 'cross_decompositio…

VERSION
    1.5.2

FILE
    /Users/ajmaljalal/.pyenv/versions/3.12.5/lib/python3.12/site-
packages/sklearn/__init__.py
```

1.0.2 Step 2.

The `sklearn` package includes some built-in datasets that can be imported. One of these is a collection of low-resolution images (8 x 8 pixels) representing hand-written digits. Let's import the dataset:

In the coded cell below:

- First import the `datasets` submodule from the sklearn package
- Next call the `load_digits()` function in the `datasets` module, and assign the result to the variable `digits`.
- Using the built-in function type, print the type of the `digits` variable.

Graded Cell This cell is worth 5% of the grade for this assignment.

[2]:
```
from sklearn import datasets
digits = datasets.load_digits()
print(type(digits))
```

```
<class 'sklearn.utils._bunch.Bunch'>
```

1.0.3 Step 3.

You should see that `digits` is an object of type 'sklearn.utils.Bunch', which is not something we have seen before, but it is basically a new type of container that is something like a Python dictionary. (One of the ways it differs from a dictionary is that elements contained in the Bunch can be accessed using the dot operator `.` rather than the square-bracket indexing supported by dictionaries. We'll see this feature below.)

Because a Bunch is similar to a dictionary, it can be queried to list its keys. Print out the result of `digits.keys()` and examine the output.

[4]:
```
print(digits.keys())
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images',
'DESCR'])
```

1.0.4 Step 4.

You should notice that `digits` contains multiple elements, one of which is `images`, which we can access via the expression `digits.images`, that is, using the dot operator to get the images out of the digits Bunch. In the code cell below:

- print the types of the items `images` and `target` contained in `digits`.
- print out the shape of both the `images` and `target` arrays.

Graded Cell This cell is worth 10% of the grade for this assignment.

[5]:
```
# Printing types of images and target
print("Type of images:", type(digits.images))
print("Type of target:", type(digits.target))

# Printing shapes of images and target arrays
print("Shape of images:", digits.images.shape)
print("Shape of target:", digits.target.shape)
```

```
Type of images: <class 'numpy.ndarray'>
Type of target: <class 'numpy.ndarray'>
```

```
Shape of images: (1797, 8, 8)
Shape of target: (1797,)
```

1.0.5 Step 5.

You should notice that `images` is a three-dimensional array of shape (1797, 8, 8) and that `target` is a one-dimensional array of shape (1797,). Each array contains 1797 elements in it, since these are 1797 examples of hand-written digits in this dataset. Let's have a look at the data in more detail.

In the code cell below: * print the value of the first image in the array * print the value of the first target

Graded Cell This cell is worth 5% of the grade for this assignment.

[6]:
```python
# Printing first image array
print("First image:")
print(digits.images[0])

# Printing first target value
print("\nFirst target:", digits.target[0])
```

```
First image:
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]

First target: 0
```

1.0.6 Step 6.

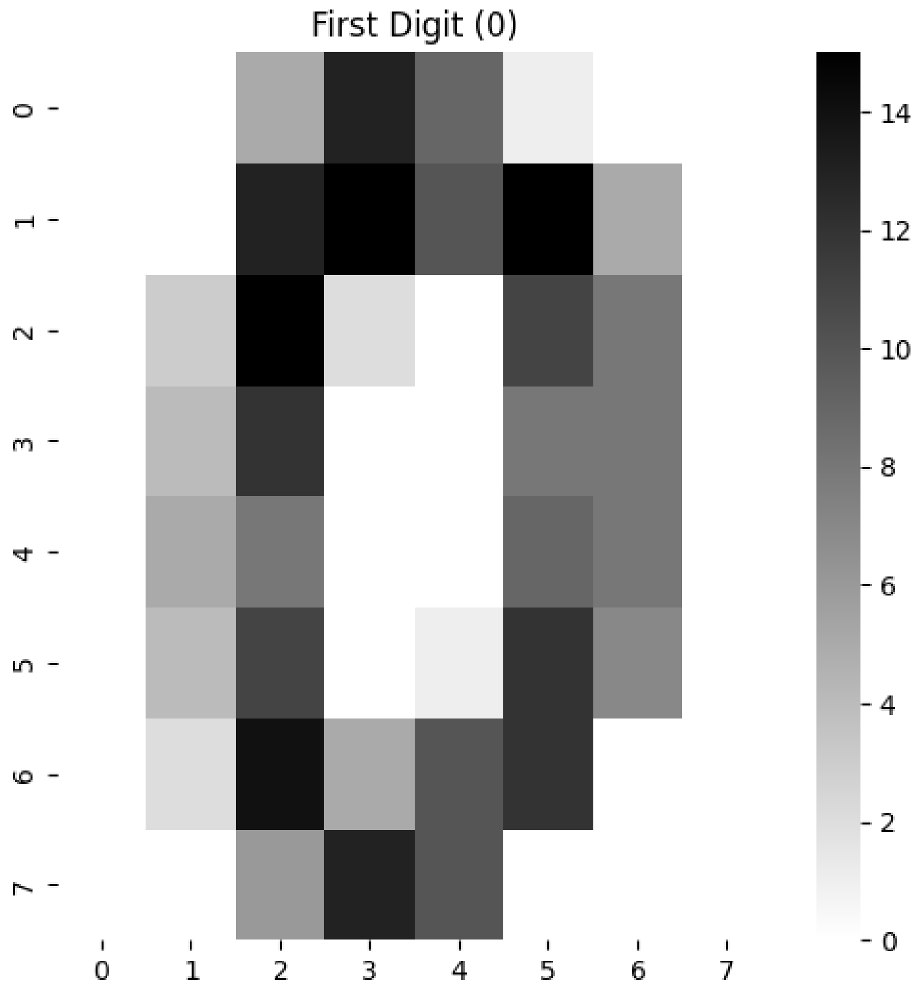
Because the images array has shape (1797, 8, 8), the first entry in that array (`digits.images[0]`) is an 8 x 8 subarray. This array encodes the grayscale value of the first hand-written image in the dataset, i.e., each entry in the 8 x 8 array encodes the intensity (darkness) of the corresponding pixel. From the output above, the value of `digits.target[0]` is reported to be `0`. This means that the first image in the dataset is an example of the digit 0. It's a bit difficult to see that by staring at the numbers in the 8 x 8 image array, but maybe things will make more sense if we try to visualize the image.

Please use the seaborn heatmap function to display the image in the code cell below. Hopefully that looks something like a zero to you.

Graded Cell This cell is worth 10% of the grade for this assignment.

[7]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Creating a heatmap of the first digit
plt.figure(figsize=(6,6))
sns.heatmap(digits.images[0], cmap='gray_r')
plt.title('First Digit (0)')
plt.show()
```



1.0.7 Step 7.

The `digits` Bunch also contains an item called `data`, which is also a numpy array. In the code cell below, print out the shape of the data item.

Graded Cell This cell is worth 5% of the grade for this assignment.

[8]:
```python
print("Shape of digits.data:", digits.data.shape)
```

```
Shape of digits.data: (1797, 64)
```

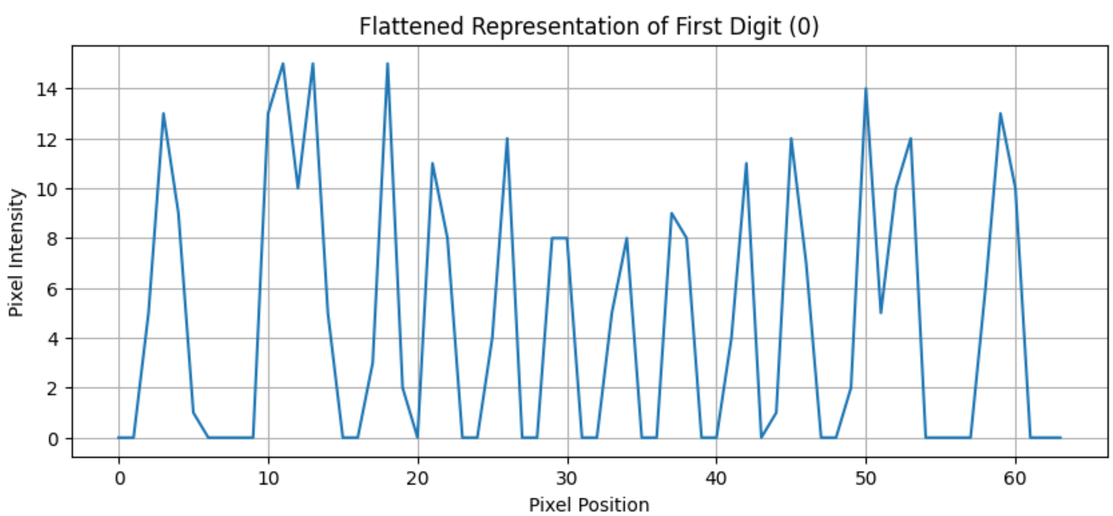
1.0.8 Step 8.

You should see that `digits.data` has shape (1797, 64). This reflects the fact that for each of the 1797 hand-written images in the dataset, the 8 x 8 image array has been "flattened" into a one-dimensional data array of length 64, by concatenating each of the 8 rows one after the other. (Within numpy, an n-dimensional array can be flattened into a one-dimensional array using the function np.ravel.) A flattening like this is convenient to be able to feed data into a machine learning algorithm, since we can use the same algorithm for datasets of different dimensions. No information is lost by this flattening procedure, except for the fact that if we were to plot out the flattened array, we probably would not be able to recognize what digit is encoded. In the code cell below, make a simple line plot using `plt.plot` of the one-dimensional data in array digits.data[0] to see what the flattened version of the data looks like.

Graded Cell This cell is worth 5% of the grade for this assignment.

[9]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.plot(digits.data[0])
plt.title('Flattened Representation of First Digit (0)')
plt.xlabel('Pixel Position')
plt.ylabel('Pixel Intensity')
plt.grid(True)
plt.show()
```



1.0.9 Step 9.

We've gone through multiple steps of interrogating the digits dataset, since this is typical in the process of developing a machine learning analysis, where one needs to understand the structure of the data and how the different data items relate to each other. We're going to carry out a supervised learning classification of the data.

In this classification process, we are going to train a classifier on labeled examples, where the labels are the known values in the target array. For example, the classifier will be instructed that the data in `digits.data[0]` corresponds to the digit 0, the data in `digits.data[314]` corresponds to the digit 6, etc.

The material in the sklearn tutorial on Learning and predicting describes this next phase of the process, which we will incorporate into the code cell below.

In the code cell below please perform the following tasks:

- Import the `svm` classifier from `sklearn`
- Creates an object of type SVC (Support Vector Classifier) and assigns it to the variable `clf` (short for classifier). Set Gamma to 0.01 and C to 100 which are hyperparameters that can be specified by the user before training. They define the classification boundary between classified and missclassified data points. We have selected some sample values for this assignment but in practice there are heuristics and cross-validation procedures to identify good values.
- Fits (trains) the data in all of the images and targets except for the last (`digits.data[:-1]`, which stops one item short of the last entry)

Graded Cell This cell is worth 10% of the grade for this assignment.

[10]:
```
from sklearn import svm

# Creating SVM classifier with specified hyperparameters
clf = svm.SVC(gamma=0.01, C=100)

# Training the classifier on all but the last example
clf.fit(digits.data[:-1], digits.target[:-1])
```

[10]:
```
SVC(C=100, gamma=0.01)
```

1.0.10 Step 10.

Having fit the classifier on all but the last image, we can now try to predict the digit associated with the last image, by calling the `predict` method on our classifier `clf`.

In the code cell below:

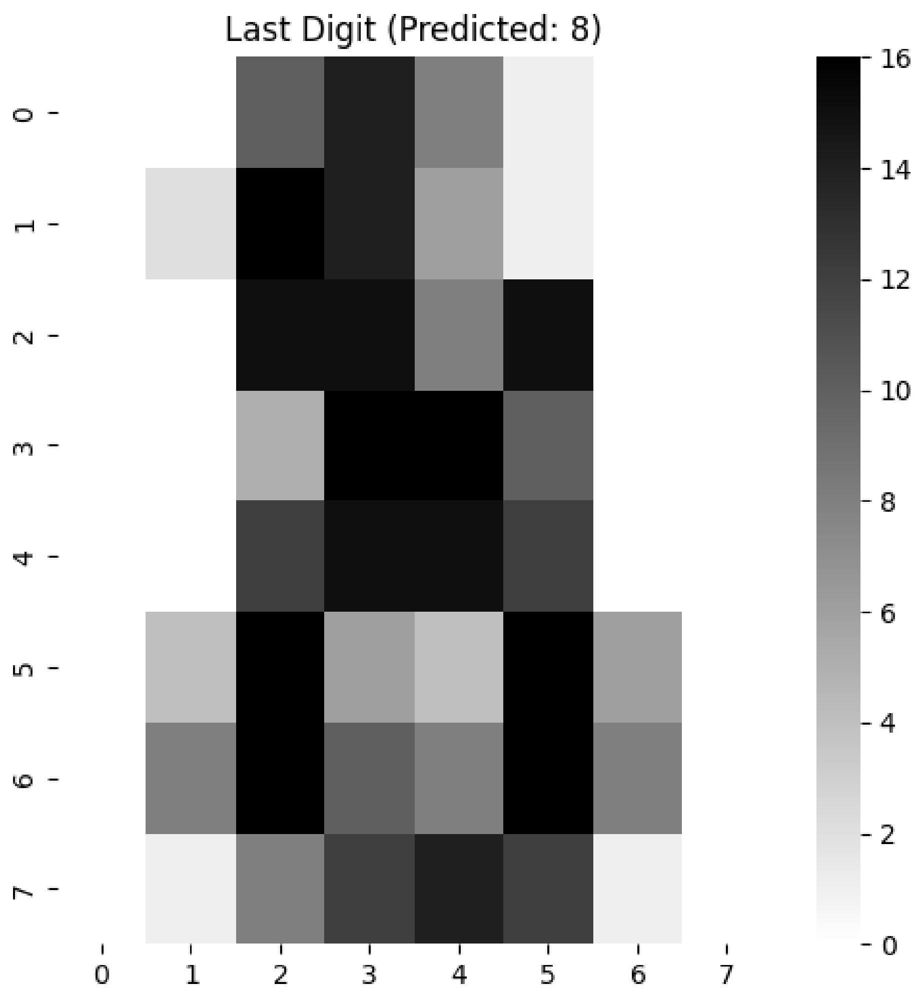
- Apply the trained model to recognize the digit in the last image and print the digit.
- Make a heatmap plot of the last image in the dataset. Does it look like the number 8? The sklearn tutorial notes: "As you can see, it is a challenging task: after all, the images are of poor resolution. Do you agree with the classifier?"

Graded Cell This cell is worth 10% of the grade for this assignment.

[11]:
```
# Predicting the digit for the last image
predicted_digit = clf.predict(digits.data[-1:])
print("Predicted digit for the last image:", predicted_digit[0])

# Visualizing the last image using a heatmap
plt.figure(figsize=(6, 6))
sns.heatmap(digits.images[-1], cmap='gray_r')
plt.title(f'Last Digit (Predicted: {predicted_digit[0]})')
plt.show()
```

```
Predicted digit for the last image: 8
```



1.0.11 Step 11.

The last digit in the dataset was *predicted* to be 8, based on the trained classifier. In the code cell below, write an expression that assigns to the variable true_last_digit the true value of the last digit in the dataset, by extracting the relevant value out of the digits object.

Graded Cell This cell is worth 20% of the grade for this assignment.

[12]:
```
true_last_digit = digits.target[-1]
print("True value of the last digit:", true_last_digit)
```

```
True value of the last digit: 8
```

1.0.12 Step 12.

In the example above, we trained the classifier using all but one example, and then tried to predict the digit for that last remaining example. That is just one of many possible workflows, using a particular split of training and testing data. For example, we could instead train on all but the last 100 examples, and then predict the last 100 examples using that model.

In the code cell below, fit the `clf` classifier on all but the last 100 examples, and then predict the digits for the last 100 examples. Save your result to the variable `predict_last_100`, and print out the value of that variable so that you can observe the set of predictions made for this test dataset.

Graded Cell This cell is worth 20% of the grade for this assignment.

[17]:
```
# Training the classifier on all but the last 100 examples
clf.fit(digits.data[:-100], digits.target[:-100])

# Predicting the digits for the last 100 examples
predict_last_100 = clf.predict(digits.data[-100:])
print("Predictions for the last 100 digits:", predict_last_100)
```

```
Predictions for the last 100 digits: [0 9 5 5 6 5 0 9 8 9 8 8 1 8 7 3 5 1 8 8 2
2 7 8 2 8 8 8 6 8 8 8 8 8 4 8 6
 8 8 9 1 5 0 9 5 8 8 2 8 0 8 7 6 8 2 8 8 8 6 3 1 3 9 1 7 6 8 4 8 1 8 8 5 3
 6 9 6 1 7 5 4 8 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8]
```