# DigitRecognition

February 21, 2023

## 1 Digit Recognition with Kernel Perceptrons and SVMs

In this exercise, you are asked to run an experimental evaluation of SVMs and the perceptron algorithm, with and without kernels, on the problem of classifying images representing digits. **This exercise is mandatory for ITCS 8156.**

The UCI Machine Learning Repository maintains datasets for a wide variety of machine learning problems. For this assignment, you are supposed to work with the Optical Recognition of Handwritten Digits Data Set. The actual dataset is located here.

### 1.1 Write Your Name Here:

## 2 Submission instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version showing the code and the output of all cells, and save it in the same folder that contains the notebook file.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly.
7. Submit **both** your PDF and the notebook file .ipynb on Canvas, as well as any module you have written that are required to run the code, such as `perceptron.py`.
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

### 2.1 1. Pre-processing the Dataset (10 points)

Read the description of the dataset, and pre-process as follows:

1. Download the training set `optdigits.tra` and the test set `optdigits.tes`. Use the first 1000 examples in `optdigits.tra` for *development* and the rest of 2823 examples for `training`. Use all 1797 examples in `optdigits.tes` for *testing*. - This was done for you, use the `devel.txt`, `train.txt`, and `test.txt` files from `../data/optdigits/`, created using the sparse SVM format.
2. Create train, devel, and test files for each of the 10 digits, setting the class to 1 for instances of that digit, and to -1 for instances of other digits, i.e. *one-vs-rest* scenario. - For

each a digit `<d>`, 3 files will be created in `../data/optdigitcs/`: the training examples in `train-<d>.txt`, the development examples in `devel-<d>.txt`, and the test examples in `test-<d>.txt`. - The examples in each of these files should appear in the same order as in the corresponding `devel.txt`, `train.txt`, and `test.txt` files.

[ ]:
```
# YOUR CODE HERE
```

### 2.2 2. Min-max Scaling (10 points)

Write functions for scaling all the features between $[0, 1]$, as discussed in class, using the *min* and *max* computed over the training examples.

All experiments in this assignment will be run on scaled data.

[ ]:
```
import numpy as np

# Compute the min and max for each feature (column)
# across the training examples (rows) from the data matrix X.
def mean_std(X):
    # YOUR CODE HERE
    min = 0
    max = 0

    return min, max


# Scale the features of the examples in X by subtracting their min and
# dividing by max - min, as provided in the parameters.
def scale(X, min, max):
    # YOUR CODE HERE
    S = X

    return S
```

### 2.3 3. Experiments with the Linear Perceptron (30 points)

Place all your perceptron and kernel perceptron functions in a module called `perceptron.py`, e,g, the training and prediction functions, all kernel implementations, as well as files for reading data, reading and writing model files.

Train first the linear perceptron, with the number of epochs set to $T \in \{1, 2, ..., 20\}$. After training each linear perceptron, normalize the learned weight vector (by dividing it by its L2 norm). Select for $T$ the value that obtains the best overall accuracy on the development data, and use this value for the remaining perceptron experiments.

For each $T$ value, you will have trained 10 models, one for each digit. In order to compute the label for a test or development example, you will run the 10 trained models on that example and output the label that obtains the highest score. Compute the accuracy on the development data

and identify the $T$ value that obtains the best accuracy. Use this tuned $T$ to compute the overall performance on the test data.

[ ]:
```
import perceptron

# YOUR CODE HERE
```

### 2.4 4. Experiments with the Kernel Perceptron (60 points)

For the kernel perceptron, experiment with polynomial kernels $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^d$ with degrees $d \in \{2, 3, 4, 5, 6\}$, and with Gaussian kernels $k(\mathbf{x}, \mathbf{y}) = exp(-||\mathbf{x} - \mathbf{y}||^2/2\sigma^2)$ with the width $\sigma \in \{0.1, 0.5, 2, 5, 10\}$. For each hyper-parameter value, you will have trained 10 models, one for each digit. In order to compute the label for a test or development example, you will run the 10 trained models and output the label that obtains the highest score. Compute the accuracy on the development data and identify the hyper-parameter value that obtains the best accuracy. Use the tuned hyper-parameter ($d$ for poly-kernel, $\sigma$ for Gaussian) to compute the overall performance on the test data.

For each of the three perceptrons (linear, poly kernel and Gaussian kernel) report the total training time, the overall accuracy, and the number of support vectors. Show and compare the corresponding 4 confusion matrices. Which digit seems to be the hardest to classify? Which perceptron / kernel combination achieves the best performance? Which algorithms are slower at training time, and why?

[ ]:
```
# YOUR CODE HERE
```

---

### 2.5 5. Experiments with Support Vector Machines (100 points)

Run the same experiments using SVMs instead of perceptrons, i.e. linear SVMs and SVMs with polynomial and Gaussian kernels. Use the same tuning scenarios for the hyper-parameters of the polynomial and Gaussian kernels. Use $C = 1$ in all SVM experiments. Report the same types of results and analysis as above, and compare with the perceptron results.

You are free to use packages with interaces in Python such as *SVMLight*, *LIBSVM*, or *Scikit-Learn*. Their web sites contain plenty of documentation on how to use them. If you use *Scikit-Learn*, the following functionality from the `sklearn.svm` will be useful:

- `SVC()`: This is the main class used for SVM classification models. Its implementation is based on *LIBSVM*. Make sure that you properly map the SVM hyper-parameters to the parameters in the constructor of this class. For example, the *gamma* parameter in the constructor corresponds to our $1/2\sigma^2$ coefficient in the Gaussian kernel. The formulas for the kernels implemented by SVC are described in this UserGuide.
- `decision_function(x)`: Once the classifier is trained, this will compute the distance between a sample `x` and the decision hyperplance. This is the quantity that you can use to determine the highest scoring class when training the 10 *one-vs-rest* classifiers: once a classifier is trained for all 10 digits, given a sample $x$ you compute this quantity for all 10 classifiers and select the class that corresponds to the classifier with largest decision function value.

- `fit()`: This is the function used to train the classifier.
- `predict(x)`: This is used to calculate the (binary) label for sample $x$.

*LIBSVM*, and therefore *Scikit-Learn* too, already implement the *one-vs-rest* classification scheme. In this scheme, you can directly use the training dataset with the 10 original labels, and *Scikit-Learn* will train the 10 binary classifiers for you. You can use this capability for this assignment, however bonus points will be given if you train the 10 binary classifiers directly, as described for the perceptron algorithm above, by creating a binary training dataset for each class.

[ ]:
```
# YOUR CODE HERE
```

### 2.6 6. Anything extra goes here

[ ]:
```

```

### 2.7 7. Analysis of results (30 points)

Include here a nicely formatted report of the results, comparisons. Include explanations and any insights you can derive from the algorithm behavoir and the results. This section is important, so make sure you address it appropriately.

[ ]:
```

```