01_assignment_perceptron_robertson

February 15, 2025

Assignment 1: Perceptron

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

Student: Hailey Robertson, hdr22

Prerequisites:

1. Enable Google Colaboratory as an app on your Google Drive account
2. Create a new Google Colab notebook, this will also create a "Colab Notebooks" directory under "MyDrive" i.e.

```
/content/drive/MyDrive/Colab Notebooks
```

3. Create the following directory structure in your Google Drive

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Assignments
```

4. Move the 01_assignment_perceptron_multiclass.ipynb into

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Assignments
```

so that its absolute path is

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Assignments/01_assignmen
```

In this assignment, you will implement both binary and multiclass perceptron classifiers from scratch. You will test your implementations on the digits dataset from scikit-learn. The assignment is divided into three main parts:

1. Implementing a binary perceptron for digit classification (0 vs 1)
2. Implementing a multiclass perceptron for full digits classification (0-9)
3. Comparing your implementations with scikit-learn's Perceptron

Submission:

1. Implement all TODOs in the code blocks below.
2. Report your validation and testing scores. For full credit, your testing scores should be higher than 0.9.

```
BinaryPerceptron:
Best model test accuracy: 0.9861
```

```
Scikit-learn Perceptron test accuracy: 1.0000

MultiClassPerceptron:
Best model test accuracy: 0.9611
Scikit-learn Perceptron test accuracy: 0.9361
```

3. List any collaborators.

```
Collaborators: N/A
```

[26]:
```
import numpy as np
import sklearn.datasets as skdata
from sklearn.linear_model import Perceptron
import sklearn.metrics as skmetrics
from sklearn.model_selection import train_test_split
import warnings, time
import matplotlib.pyplot as plt

warnings.filterwarnings(action='ignore')
np.random.seed(42)
```

[27]:
```
class BinaryPerceptron:
    '''
    Implementation of Binary Perceptron
    '''

    def __init__(self):
        self.__weights = None

    def __update(self, x, y):
        '''
        Update weights for misclassified examples

        Arg(s):
            x : numpy.ndarray
                Feature vector of shape d x 1
            y : int
                Label/target (-1 or 1)
        '''

        # DONE: Implement weight update rule for binary perceptron
        # NOTE: x = (d+1, 1) column vector with bias, so select first column␣
 ↪for (d+1,) array
        self.__weights[:, 0] += y * x

    def fit(self, x, y, max_iter=100):
        '''
        Fit the binary perceptron to training data
```

```
        Arg(s):
            x : numpy.ndarray
                Features of shape d x N
            y : numpy.ndarray
                Labels/targets of shape (1 x N) or (N x 1)
            max_iter : int
                Maximum number of iterations
        '''

        n_features, n_samples = x.shape

        # DONE: Initialize weights (including a bias term, w0) as zeros vector␣
↪with shape d+1 x 1
        self.__weights = np.zeros((n_features + 1, 1))

        # DONE: Append artificial coordinate (x0) to the data
        # NOTE: This makes the shape d+1 x 1 since bias isn't separate
        x = np.concatenate((np.ones((1, n_samples)), x), axis=0)

        # DONE: Implement training loop
        for _ in range(max_iter):
            n_updates = 0

            # Process each sample
            for n in range(n_samples):

                # DONE: Calculate prediction
                binary_prediction = np.sign(self.__weights.T @ x[:, n])
                # print(binary_prediction, y[n]) # shape (1,1) but single value

                # DONE: Update weights if misclassified
                if binary_prediction != y[n]:
                    self.__update(x[:, n], y[n])
                    # print(self.__weights)
                    n_updates += 1

            # DONE: Break if no updates were made, e.g., check for convergence
            if n_updates == 0:
                break

    def predict(self, x):
        '''
        Make predictions

        Arg(s):
            x : numpy.ndarray
```

```
                Features of shape d x N

        Returns:
            numpy.ndarray : Predicted labels (-1 or 1) of 1 x N
        '''

        n_features, n_samples = x.shape

        # DONE: Append artificial coordinate (x0) to the data
        x = np.concatenate((np.ones((1, n_samples)), x), axis=0)

        # DONE: Implement prediction logic
        # NOTE: Was having a lot of trouble getting shapes to match (see notes␣
↪above) so ended up doing .flatten() to deal with scalar
        binary_prediction = np.sign(self.__weights.T @ x).flatten()
        return binary_prediction

    def score(self, x, y):
        '''
        Calculate prediction accuracy

        Arg(s):
            x : numpy.ndarray
                Features of shape d x N
            y : numpy.ndarray
                Labels/targets of shape (1 x N) or (N x 1)

        Returns:
            float: Accuracy score
        '''

        # DONE: Implement accuracy calculation
        binary_predictions = self.predict(x)
        binary_accuracy = np.mean(binary_predictions == y.flatten())
        return binary_accuracy
```

[ ]:
```
class MulticlassPerceptron:
    '''
    Implementation of Multiclass Perceptron using one-vs-rest strategy
    '''

    def __init__(self):
        self.__weights = None
        self.__n_classes = None

    def __update(self, x, y, y_hat):
        '''
```

```
        Update weights for misclassified examples
s
        Arg(s):
            x : numpy.ndarray
                Feature vector of shape d x 1
            y : int
                Label/target (-1 or 1)
            y_hat : int
                Predicted label (-1 or 1)
        '''

        # DONE: Implement weight update rule for multiclass case
        self.__weights[:, y] += x
        self.__weights[:, y_hat] -= x

    def fit(self, x, y, max_iter=100):
        '''
        Fit the multiclass perceptron to training data

        Arg(s):
            x : numpy.ndarray
                Feature vector of shape d x N
            y : numpy.ndarray
                Label/target (-1 or 1) of shape (1 x N) or (N x 1)
            max_iter : int
                Maximum number of iterations
        '''

        # Flattening y for 1D everywhere
        y = y.flatten()

        n_features, n_samples = x.shape

        # DONE: Get number of classes from unique values in y
        self.__n_classes = np.unique(y).size

        # DONE: Initialize weights matrix of zeros with shape d+1 x C
        self.__weights = np.zeros((n_features + 1, self.__n_classes))

        # DONE: Append artificial coordinate (x0) to the data such that it is␣
 ↪d+1 x N
        x = np.concatenate((np.ones((1, n_samples)), x), axis=0)

        # DONE: Implement training loop
        for _ in range(max_iter):
            n_updates = 0
```

```
            # Process each sample
            for n in range(n_samples):

                # DONE: Calculate scores and make prediction for each class
                scores = self.__weights.T @ x[:, n]
                y_hat = np.argmax(scores)

                # Update if prediction is wrong
                if y_hat != y[n]:
                    self.__update(x[:, n], y[n], y_hat)
                    n_updates += 1

            # DONE: Break if no updates were made, e.g., check for convergence
            if n_updates == 0:
                break

    def predict(self, x):
        '''
        Make predictions on new data

        Arg(s):
            x : numpy.ndarray
                Features of shape d x N

        Returns:
            numpy.ndarray : Predicted class labels
        '''

        n_features, n_samples = x.shape

        # DONE: Append artificial coordinate (x0) to the data
        x = np.concatenate((np.ones((1, n_samples)), x), axis=0)

        # DONE: Implement prediction logic for multiclass case
        scores = self.__weights.T @ x
        multi_prediction = np.argmax(scores, axis=0).flatten()

        return multi_prediction

    def score(self, x, y):
        '''
        Calculate prediction accuracy

        Arg(s):
            x : numpy.ndarray
                Features of shape d x N
            y : numpy.ndarray
```

```
            Label/target (-1 or 1) of shape (1 x N) or (N x 1)

        Returns:
            float : Accuracy score
        '''

        # DONE: Implement accuracy calculation
        multi_predictions = self.predict(x)
        multi_accuracy = np.mean(multi_predictions == y.flatten())  # Compare␣
↪with flattened y

        return multi_accuracy
```

[29]:
```
def prepare_binary_digits_data(digits_zero=0, digits_one=1):
    '''
    Prepare binary classification dataset from digits

    Args:
        digits_zero : int
            First digit to classify
        digits_one : int
            Second digit to classify

    Returns:
        tuple: (X_train, y_train, X_val, y_val, X_test, y_test)
            X_train : N x d
            y_train : N x 1
            X_val : M x d
            y_val : M x 1
            X_test : P x d
            y_test : P x 1
    '''

    # Load digits dataset using sklearn.datasets
    digits = skdata.load_digits()

    # Select only the two specified digits
    mask = np.isin(digits.target, [digits_zero, digits_one])
    X = digits.data[mask]
    y = digits.target[mask]

    # Convert labels to -1/1
    y = np.where(y == digits_zero, -1, 1)

    # Split into train (60%), validation (20%), and test (20%) sets using␣
↪random_state=42
```

```
    X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,␣
↪test_size=0.25, random_state=42)

    # Had to change a little to get it to work with the rest of the code (X.T␣
↪instead of X)
    return X_train.T, np.expand_dims(y_train, axis=-1), X_val.T, np.
↪expand_dims(y_val, axis=-1), X_test.T, np.expand_dims(y_test, axis=-1)
```

[30]:
```
def prepare_multiclass_digits_data():
    '''
    Prepare multiclass classification dataset from digits

    Returns:
        tuple: (X_train, y_train, X_val, y_val, X_test, y_test)
            X_train : N x d
            y_train : N x 1
            X_val : M x d
            y_val : M x 1
            X_test : P x d
            y_test : P x 1
    '''

    # Load digits dataset using sklearn.datasets
    digits = skdata.load_digits()
    X, y = digits.data, digits.target

    # Split into train (60%), validation (20%), and test (20%) sets with␣
↪random_state=42
    X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,␣
↪test_size=0.25, random_state=42)

    # NOTE: Had to change a little to get it to work with the rest of the code␣
↪(X.T instead of X so shape is (d, N))
    return X_train.T, np.expand_dims(y_train, axis=-1), X_val.T, np.
↪expand_dims(y_val, axis=-1), X_test.T, np.expand_dims(y_test, axis=-1)
```

[ ]:
```
# Binary classification experiment
print("Binary Classification Experiment (0 vs 1)")
print("-" * 50)

labels = [0, 1]
```

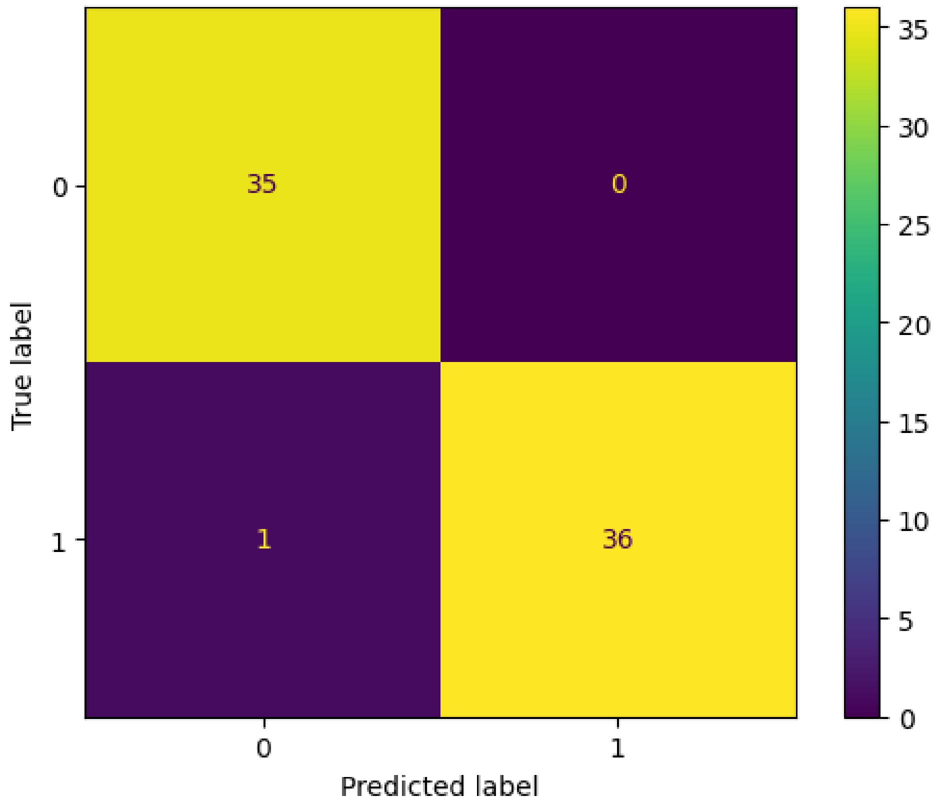
```
# Load and prepare binary data (0 vs 1)
X_train, y_train, X_val, y_val, X_test, y_test = prepare_binary_digits_data(0,␣
↪1)

# Try different max_iter values
max_iters = [10, 50, 100]
best_val_score = 0
best_model = None

for max_iter in max_iters:

    # DONE: Initialize and train binary perceptron
    model = BinaryPerceptron()
    model.fit(X_train, y_train)

    # DONE: Calculate validation score
    val_score = model.score(X_val, y_val)
    print("Max iterations: {}, Validation accuracy: {:.4f}".format(max_iter,␣
↪val_score))

    # DONE: Update best_model if current model performs better
    if val_score > best_val_score:
        best_val_score = val_score
        best_model = model

# DONE: Test best model on test set
test_score = best_model.score(X_test, y_test)
print("\nBest model test accuracy: {:.4f}".format(test_score))

# DONE: Create a confusion matrix using skmetrics.confusion_matrix on the test␣
↪set
# NOTE: Flattening since y_test is (P, 1)
binary_conf_matrix = skmetrics.confusion_matrix(y_test.flatten(), best_model.
↪predict(X_test))

# Show confusion matrix
binary_conf_matrix_plot = skmetrics.ConfusionMatrixDisplay(
    confusion_matrix=binary_conf_matrix,
    display_labels=labels
)
binary_conf_matrix_plot.plot()
plt.show()
time.sleep(1)

# DONE: Compare with scikit-learn implementation by training with max_iter=10␣
↪and random_state=42 and testing on the test set
```

```
# NOTE: Had to transpose X_train and X_test / flatten y_train and y_test to get␣
↪the shapes to match to match sk docs
sk_model = Perceptron(max_iter=10, random_state=42)
sk_model.fit(X_train.T, y_train.flatten())
sk_score = sk_model.score(X_test.T, y_test.flatten())
print("Scikit-learn Perceptron test accuracy: {:.4f}".format(sk_score))

# DONE: Create a confusion matrix using skmetrics.confusion_matrix for scikit␣
↪model on the test set
sk_binary_conf_matrix = skmetrics.confusion_matrix(y_test.flatten(), sk_model.
↪predict(X_test.T))
sk_binary_conf_matrix_plot = skmetrics.ConfusionMatrixDisplay(
    confusion_matrix=sk_binary_conf_matrix,
    display_labels=labels
)
sk_binary_conf_matrix_plot.plot()
plt.show()
time.sleep(1)


print("\nMulticlass Classification Experiment (0-9)")
print("-" * 50)

labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Load and prepare multiclass data
X_train, y_train, X_val, y_val, X_test, y_test =␣
↪prepare_multiclass_digits_data()

# Try different max_iter values
max_iters = [10, 50, 100]
best_val_score = 0
best_model = None

for max_iter in max_iters:

    # DONE: Initialize and train multiclass perceptron
    model = MulticlassPerceptron()
    model.fit(X_train, y_train)

    # DONE: Calculate validation score
    val_score = model.score(X_val, y_val)
    print("Max iterations: {}, Validation accuracy: {:.4f}".format(max_iter,␣
↪val_score))

    # DONE: Update best_model if current model performs better
    if val_score > best_val_score:
```

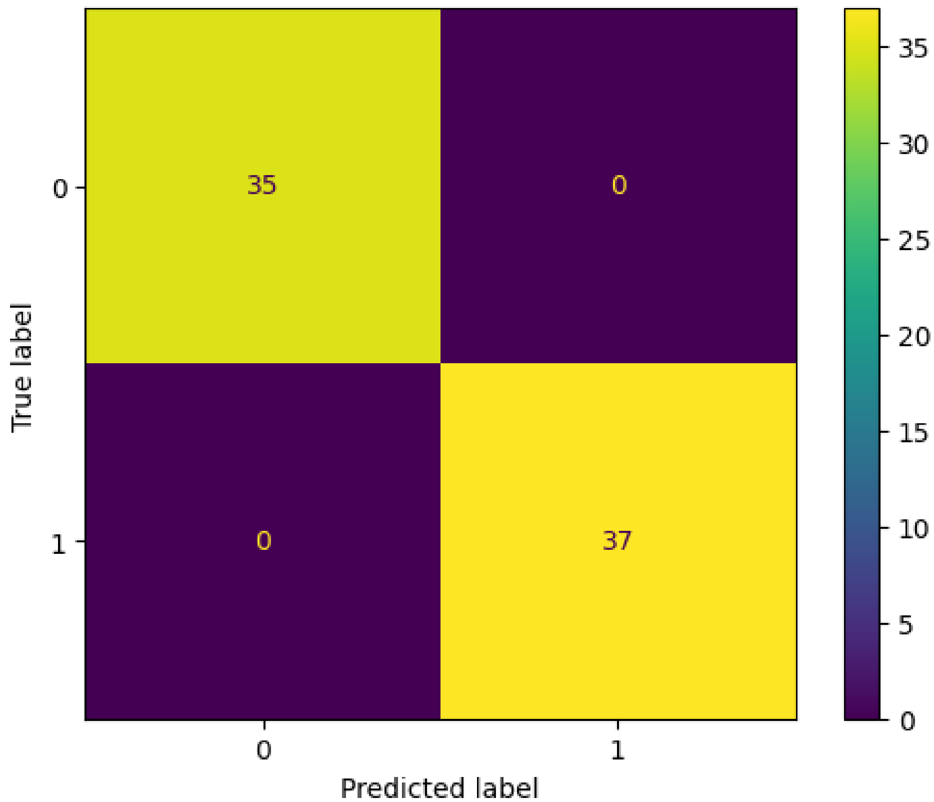
```
        best_val_score = val_score
        best_model = model

# DONE: Test best model on test set
test_score = best_model.score(X_test, y_test)
print("\nBest model test accuracy: {:.4f}".format(test_score))

# DONE: Create a confusion matrix using skmetrics.confusion_matrix for your␣
 ↪model on the test set
multi_conf_matrix = skmetrics.confusion_matrix(y_test.flatten(), best_model.
 ↪predict(X_test))
multi_conf_matrix_plot = skmetrics.ConfusionMatrixDisplay(
    confusion_matrix=multi_conf_matrix,
    display_labels=labels
)
multi_conf_matrix_plot.plot()
plt.show()
time.sleep(1)

# DONE: Compare with scikit-learn implementation by training with max_iter=10␣
 ↪and random_state=42 and testing on the test set
# NOTE: Had to transpose X_test and X_train / flatten y_train and y_test to get␣
 ↪the shapes to match to match sk docs
sk_model = Perceptron(max_iter=10, random_state=42)

sk_model.fit(X_train.T, y_train.flatten())
sk_score = sk_model.score(X_test.T, y_test.flatten())
print("Scikit-learn Perceptron test accuracy: {:.4f}".format(sk_score))

# DONE: Create a confusion matrix using skmetrics.confusion_matrix for your␣
 ↪model on the test set
sk_multi_conf_matrix = skmetrics.confusion_matrix(y_test.flatten(), sk_model.
 ↪predict(X_test.T))

# Show confusion matrix
sk_multi_conf_matrix_plot = skmetrics.ConfusionMatrixDisplay(
    confusion_matrix=sk_multi_conf_matrix,
    display_labels=labels
)
sk_multi_conf_matrix_plot.plot()
plt.show()
time.sleep(1)
```

```
Binary Classification Experiment (0 vs 1)
--------------------------------------------------
Max iterations: 10, Validation accuracy: 1.0000
Max iterations: 50, Validation accuracy: 1.0000
```

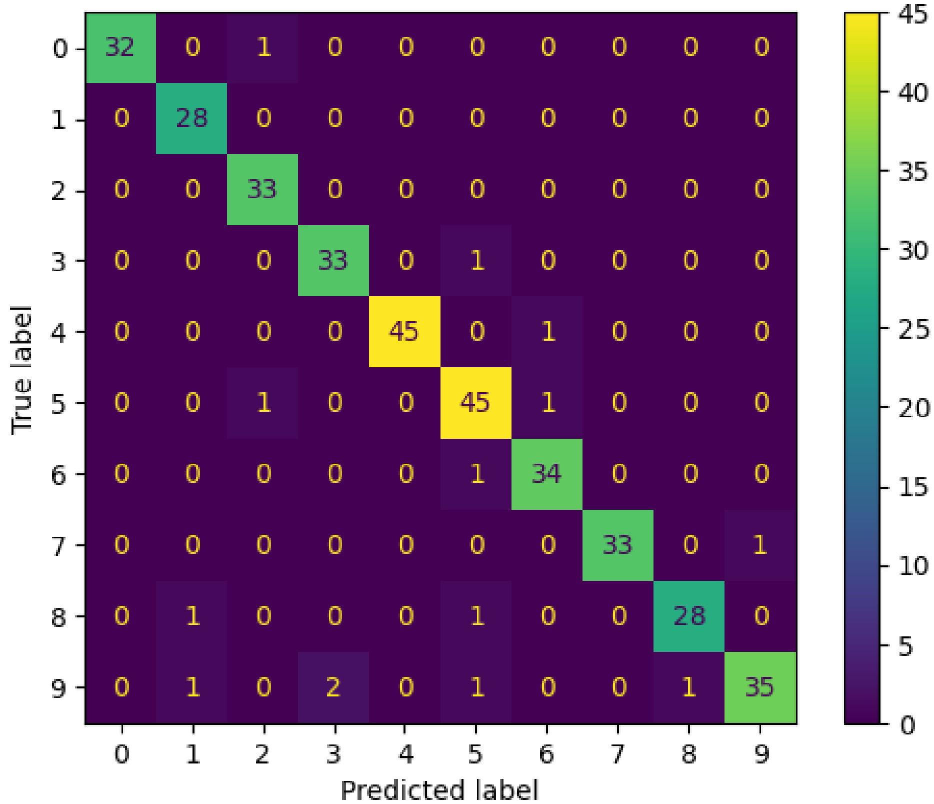
```
Max iterations: 100, Validation accuracy: 1.0000

Best model test accuracy: 0.9861
```



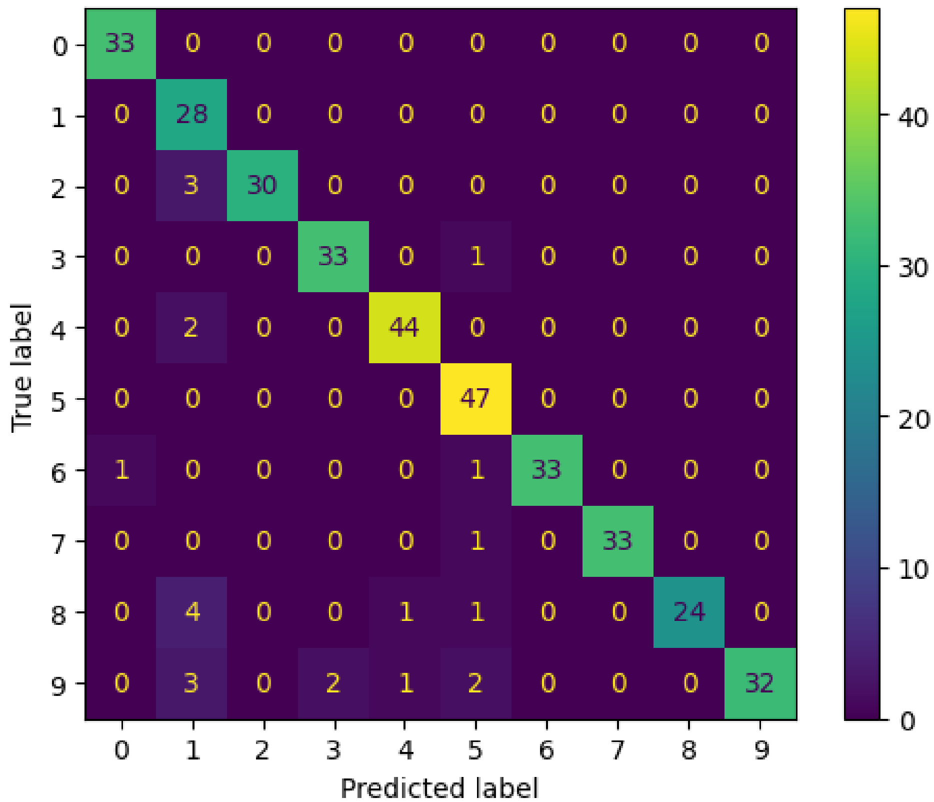

```
Scikit-learn Perceptron test accuracy: 1.0000
```




```
Multiclass Classification Experiment (0-9)
--------------------------------------------------
Max iterations: 10, Validation accuracy: 0.9583
Max iterations: 50, Validation accuracy: 0.9583
Max iterations: 100, Validation accuracy: 0.9583

Best model test accuracy: 0.9611
```




```
Scikit-learn Perceptron test accuracy: 0.9361
```


True label
Predicted label

```
[ ]:
```

```
[ ]:
```