520 Intro to AI Fall 2020 - Final Project

Po Yuan Huang - RUID 195003420, Abhishek Bhatt - RUID 198009864

April 27, 2021

1 Introduction

In this project, we design multiple classifiers: a Naive Bayes classifier, a Perceptron, a softmax regressor and a neural network. We test these classifiers on two image data sets: a set of scanned handwritten digit images (output is numerical digits from 0 to 9) and a set of face images in which edges have already been detected (output is numerical digits 1 or 0, corresponding to whether the edge image is a face or not respectively).

2 Model 1 : Naive Bayes

Features : all pixels - 1 for value, 0 for empty

Algorithm :
Training -

(a) Count number of training examples, n

(b) For labels k = 1 to K, count number of times label = k in the training set, c(y=k)

(c) For labels k = 1 to K, get probability of label = k, p(y=k) = c(y=k) / n

(d) For each of the features f = 1 to d, count number of training examples with value = u, given label is k, c(x_d=u | y=k)

(e) For each of the features f = 1 to d, get probability of feature value = u, given label is k, p(x_d=u | y=k) = c(x_d=u | y=k) / n

Validation/Testing -

(a) For labels k = 1 to K, using Naive Bayes assumption, get probability of observing a given data point $\vec{x}$ in validation/test set with feature values (v_1...v_d) as

$p(\vec{x}|y = k) = \prod_{j=1}^{d} p(x_j = v_j|y = k)$

where $p(x_j = v_j|y = k)$ has been computed during training

(b) For labels k = 1 to K, compute $p(y = k|\vec{x}) = p(\vec{x}|y = k)p(y = k)$
where $p(y = k)$ has been computed during training

(c) Classify as label k such that
$argmax_{k \in \{1,...,K\}} log(p(y = k)p(\vec{x}|y = k))$
$= argmax_{k \in \{1,...,K\}}(log(P(y = k)) + \sum_{j=1}^{d} log(p(x_{\mathrm{j}} = v_{\mathrm{j}}|y = k))$

Results :
Faces - Please refer Figure 1.
Digits - Please refer Figure 2.
Mean and Standard deviation are calculated under 4 iterations, with the default smoothing parameter value of 0.5.

3 Model 2 : Perceptron

Features : all pixels - 1 for value, 0 for empty

Algorithm :
Training -

(a) Initialize the weight vector $\vec{w} = (w_0, w_1 \ldots w_d)$

(b) For each training data point $(\vec{x^i}, y^i)$, compute
$score(\vec{w}, \vec{x^i}) = w_0 + w_1 {x_1}^i + \ldots + w_d {x_d}^i$

(c) If $score(\vec{w}, \vec{x^i}) < 0$ and y^i = true, update the weight vector as
$w_0 = w_0 + 1$
$w_j = w_j + {x_j}^i$ for j = 1 to d

(d) If $score(\vec{w}, \vec{x^i}) >= 0$ and y^i = false, update the weight vector as
$w_0 = w_0 - 1$
$w_j = w_j - {x_j}^i$ for j = 1 to d

(e) Repeat passes over the training data and weight updates for a pre-defined number of iterations (or until convergence)

(f) For labels k = 1 to K, train a perceptron each such that true implies y=k.

Validation/Testing -

(a) For a given data point $\vec{x}$ in validation/test compute
$score(\vec{w_k}, \vec{x}) = w_{k0} + w_{k1}x_1 + \ldots + w_{kd}x_d$
where k = 1 to K corresponds the perceptron learnt for each of the label classes.

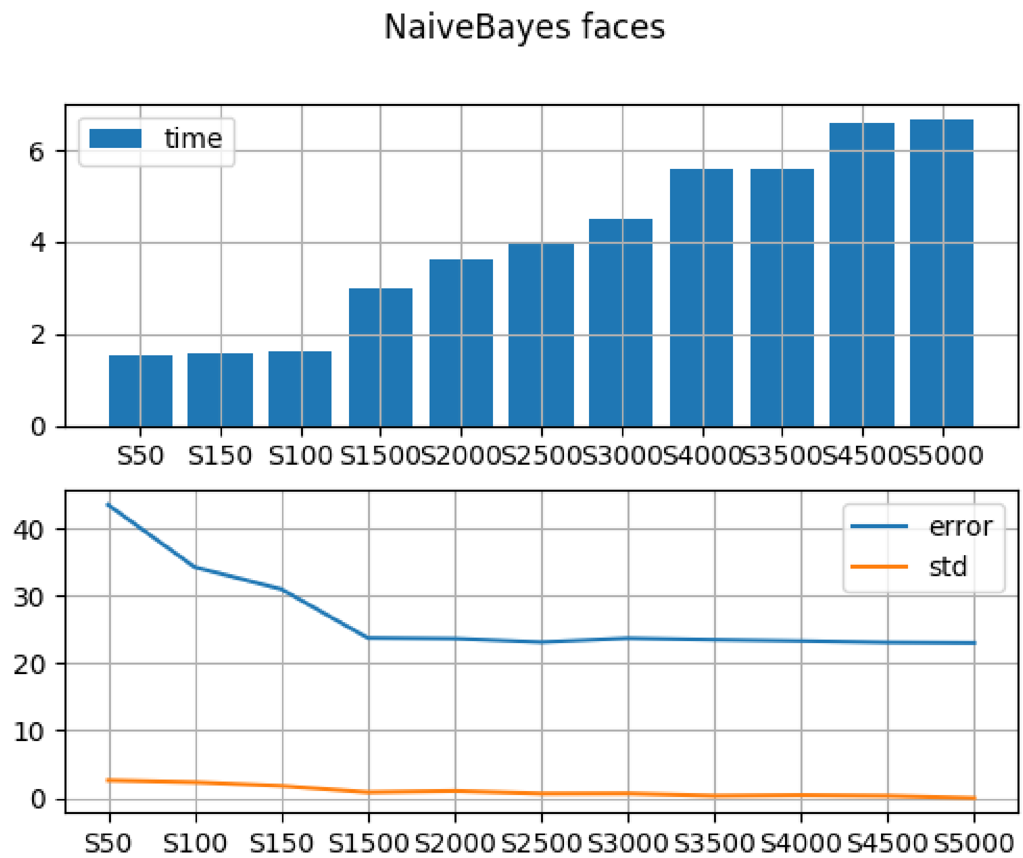

Figure 1: Faces dataset - training time, mean test error and standard deviation of test error as a function of the number of data points used for training

(b) Classify as label k such that
$argmax_{k\in\{1,\ldots,K\}} score(\vec{w_k}, \vec{x})$

Results :
Faces - Please refer Figure 3.
Digits - Please refer Figure 4.
Mean and Standard deviation are calculated under 4 iterations.

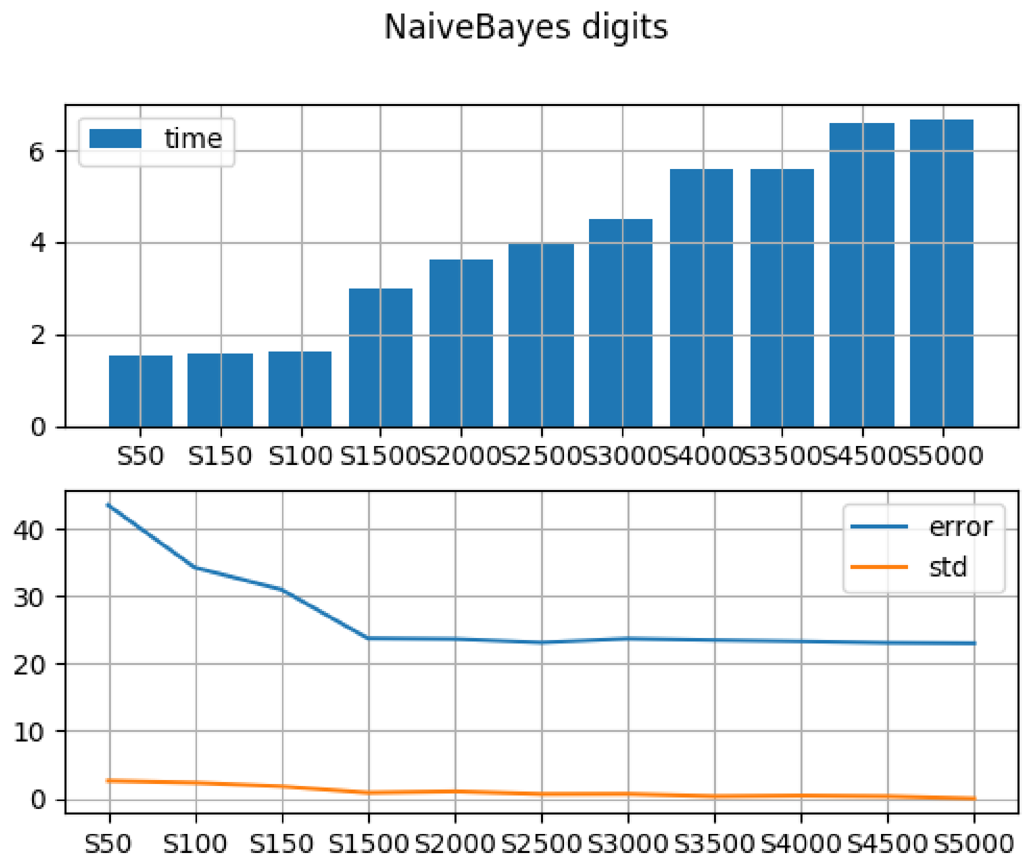

Figure 2: Digits dataset - training time, mean test error and standard deviation of test error as a function of the number of data points used for training

4 Model 3 : Softmax Regression (general version of Logistic Regression for multi-class classification)

Features / Pre-processing :

(a) faces - all pixels, set 0.99999 for value, 0.00005 for empty
digits - all pixels, set 0.99999 for value #, 0.88888 for value +, 0.00005 for empty

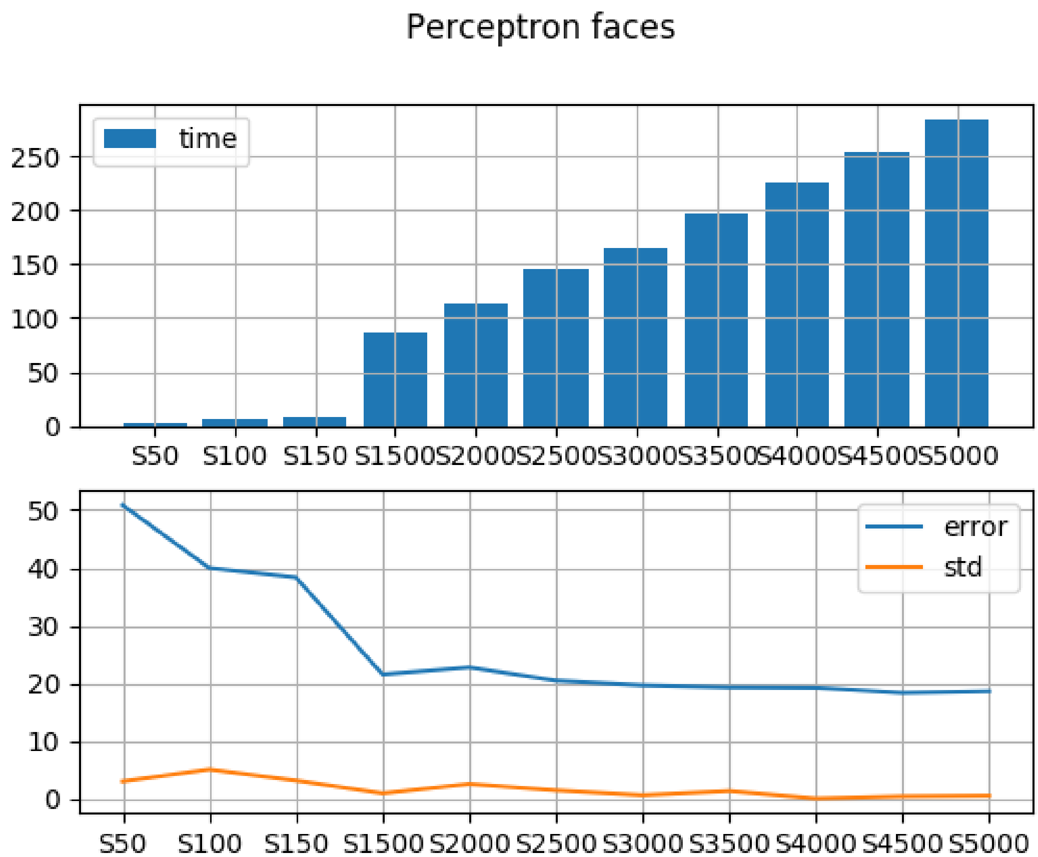

Figure 3: Faces dataset - training time, mean test error and standard deviation of test error as a function of the number of data points used for training

(b) faces - row wise moving average, with moving window size 6 and padded for maintaining image dimension
digits - row wise moving average, with moving window size 4 and padded for maintaining image dimension

(c) feature matrix flattened to vector, and label converted to one-hot encoded vector

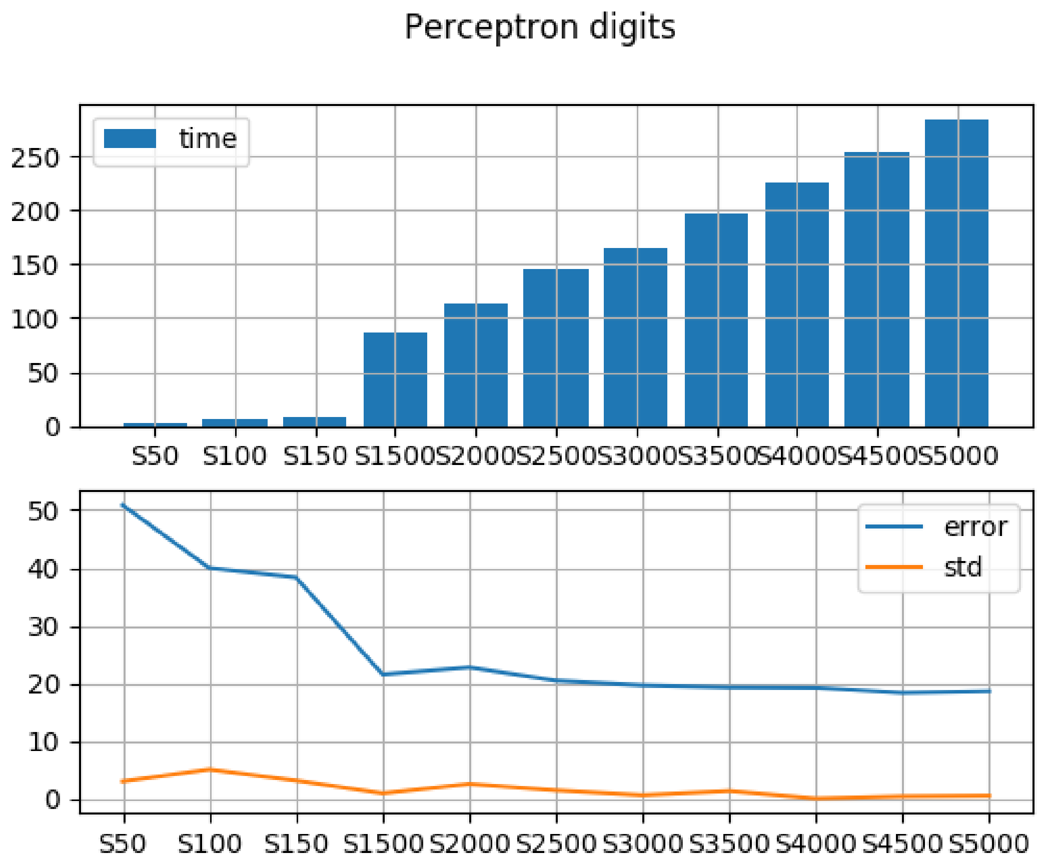

Figure 4: Digits dataset - training time, mean test error and standard deviation of test error as a function of the number of data points used for training

Cost function :

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)\top} x^{(i)})}\right]$$

Implementation with Numpy -
$Z = np.dot(theta.T, X)$
$expZ = np.exp(Z - np.max(Z))$
$A = expZ/expZ.sum(axis = 0, keepdims = True)$
$cost = (-1.0/m) * np.nansum(Y * np.log(A))$

where each column in matrix X corresponds to one training example, and each row in matrices Z and A corresponds to probability of one of the labels.

Gradient of cost function :

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right]$$

Implementation with Numpy -
$dZ = A - Y$
$dtheta = 1./m * np.dot(dZ, X.T)$

where each column in matrix X corresponds to one training example, and each row in matrices A (predicted) and Y (truth) corresponds to probability of one of the labels.

Hyperparameters :

(a) Initialization for gradient descent : Random

(b) Learning rate for gradient descent : 0.007

(c) Number of epochs (passes over the training set) for gradient descent : 1000

Results :
Faces - Please refer Figures 5, 6 and 7.
Digits - Please refer Figures 8, 9 and 10.
Mean and Standard deviation are calculated under 5 iterations.

5 Model 4 : Neural Network

Features / Pre-processing :

(a) faces - all pixels, set 0.99999 for value, 0.00005 for empty
digits - all pixels, set 0.99999 for value #, 0.88888 for value +, 0.00005 for empty

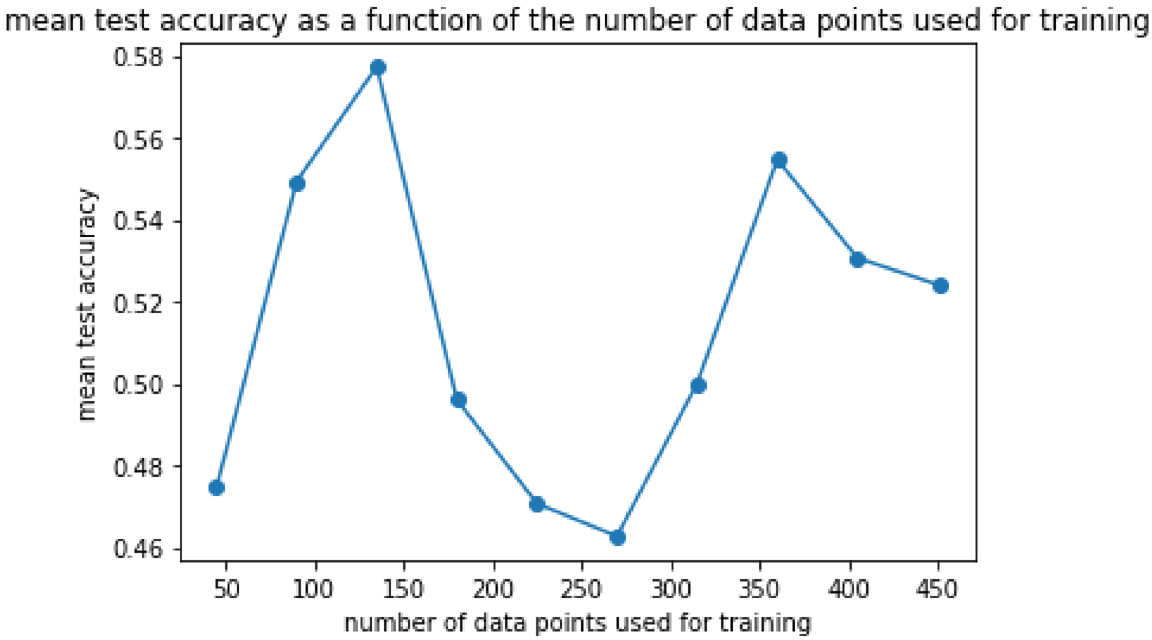

Figure 5: Faces dataset - mean test accuracy as a function of the number of data points used for training

(b) faces - row wise moving average, with moving window size 6 and padded for maintaining image dimension
digits - row wise moving average, with moving window size 4 and padded for maintaining image dimension

(c) feature matrix flattened to vector, and label converted to one-hot encoded vector

Setup and Hyperparameters :

(a) Optimizer : Adam
Exponential decay hyperparameter for the first moment estimates, beta1 = 0.9
Exponential decay hyperparameter for the second moment estimates, beta2 = 0.999
Hyperparameter preventing division by zero in Adam updates, epsilon = 1e-8

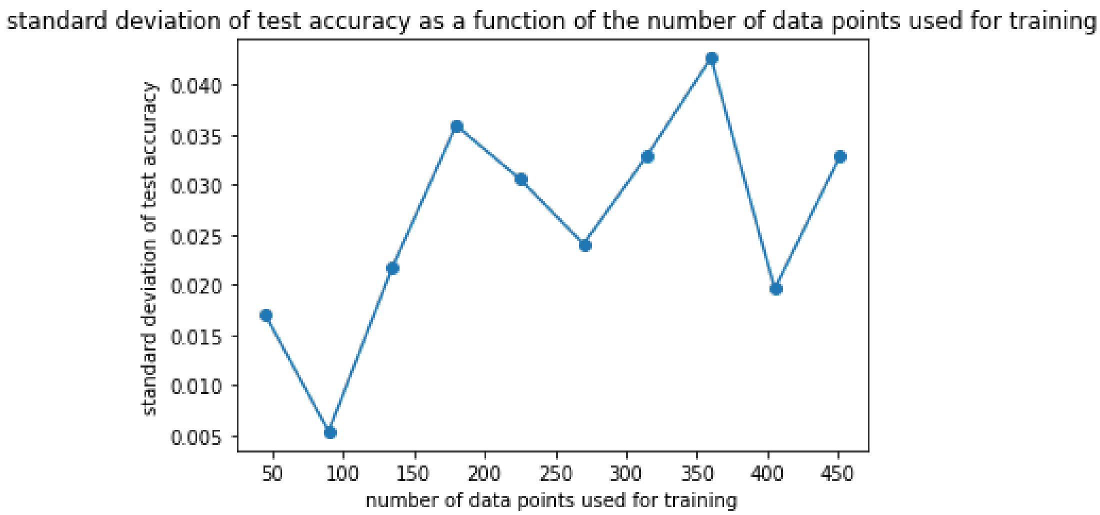

Figure 6: Faces dataset - standard deviation of test accuracy as a function of the number of data points used for training

(b) Initializer for Adam : He initialization

(c) Number of hidden units -
Input layer / features : 4200 for faces, 784 for digits
Hidden layers : 80, 16
Output layer / Softmax : 2 for faces, 10 for digits

(d) Activation function : ReLU

(e) Learning rate for gradient descent : 0.00007

(f) Number of epochs (passes over the training set) for optimization : 100

(g) Probability of keeping a hidden unit active during drop-out, keepprob = 1 (No dropout)

Results :
Faces - Please refer Figures 11, 12 and 13.
Digits - Please refer Figures 14, 15 and 16.
Mean and Standard deviation are calculated under 5 iterations.

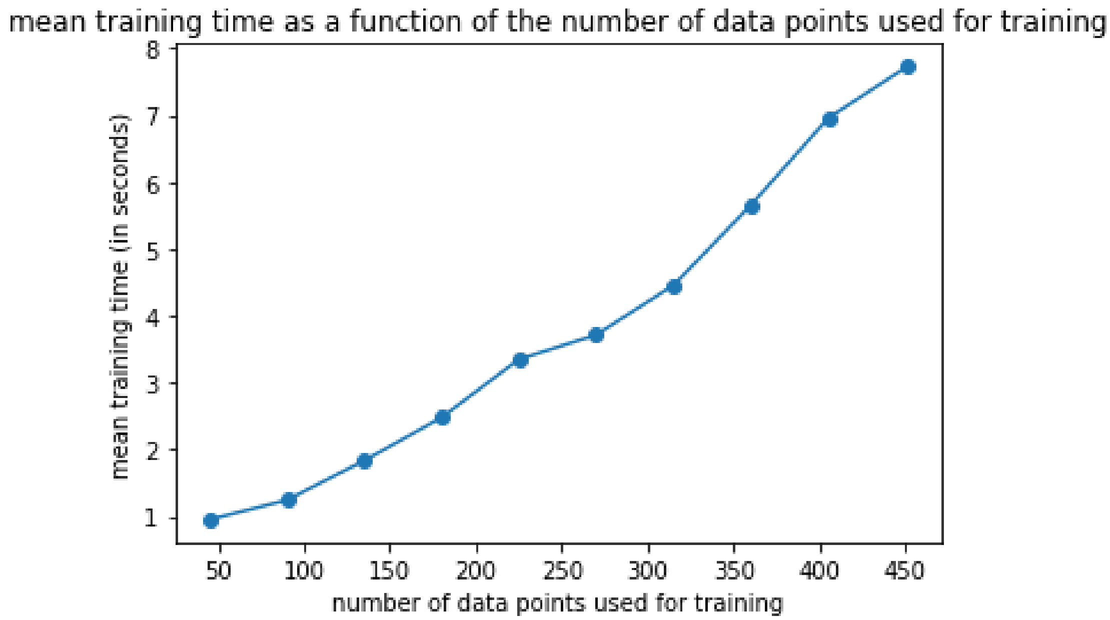

Figure 7: Faces dataset - training time as a function of the number of data points used for training

6 References and Acknowledgements

(a) Naive Bayes and Perceptron : Code architecture and utility functions used from the project created by Dan Klein and John DeNero that was given as part of the programming assignments of Berkeley CS188 course.
URL - https://inst.eecs.berkeley.edu/ cs188/sp11/projects/classification/classification.html

(b) Softmax Regression :
URL - http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/

(c) Neural Network : Code architecture and utility functions used from learnings from multiple course assignments in the Deep Learning Specialization by deeplearning.ai.
URL - https://www.coursera.org/specializations/deep-learning

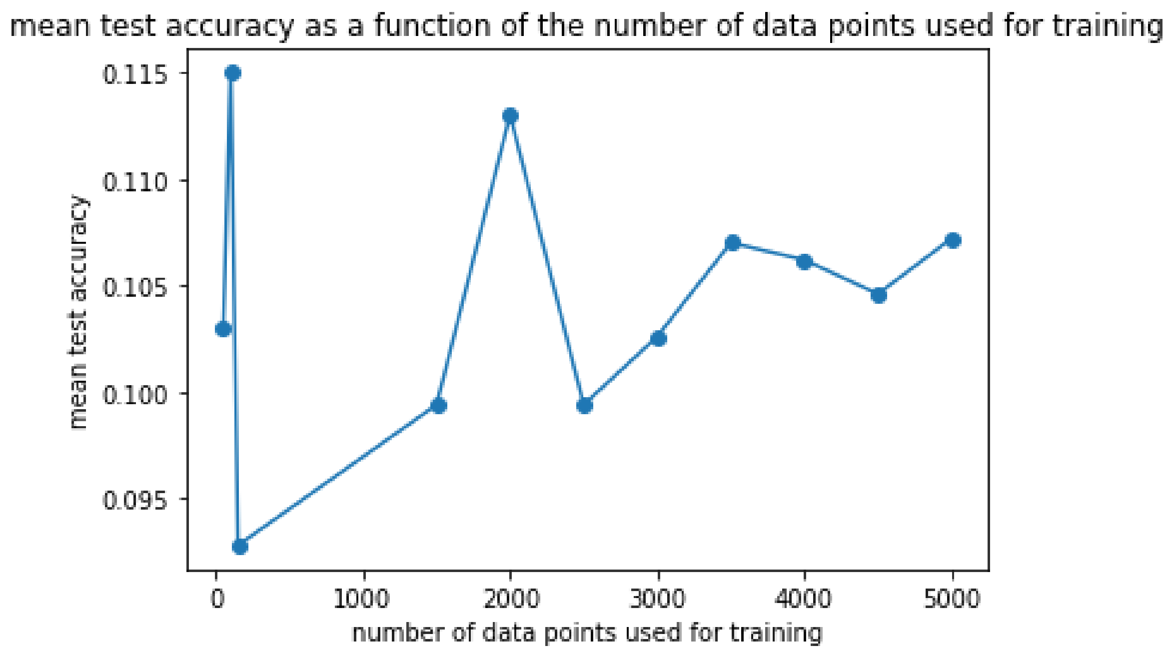

Figure 8: Digits dataset - mean test accuracy as a function of the number of data points used for training

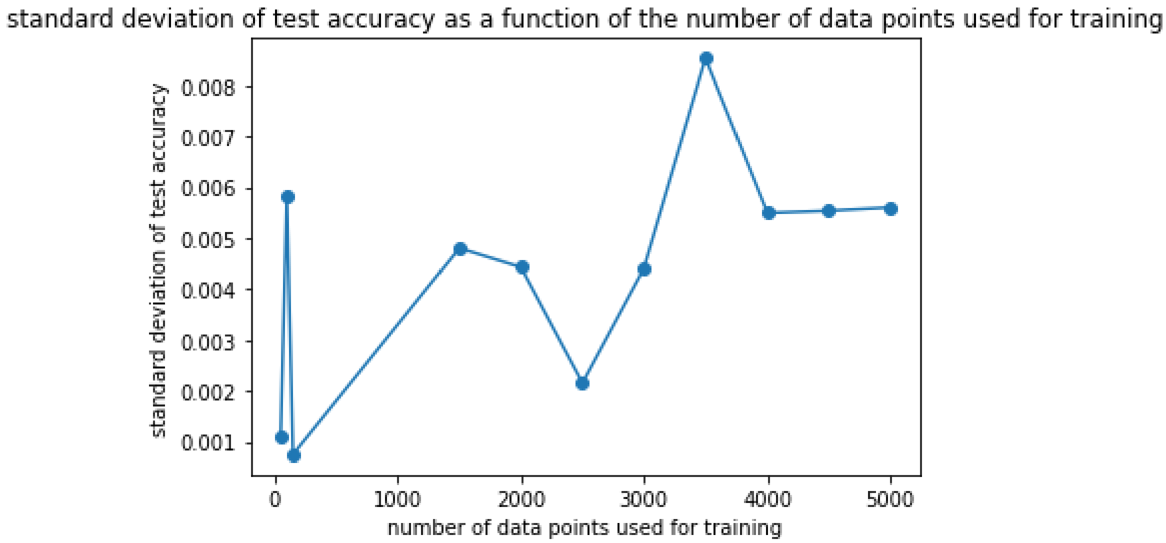

Figure 9: Digits dataset - standard deviation of test accuracy as a function of the number of data points used for training

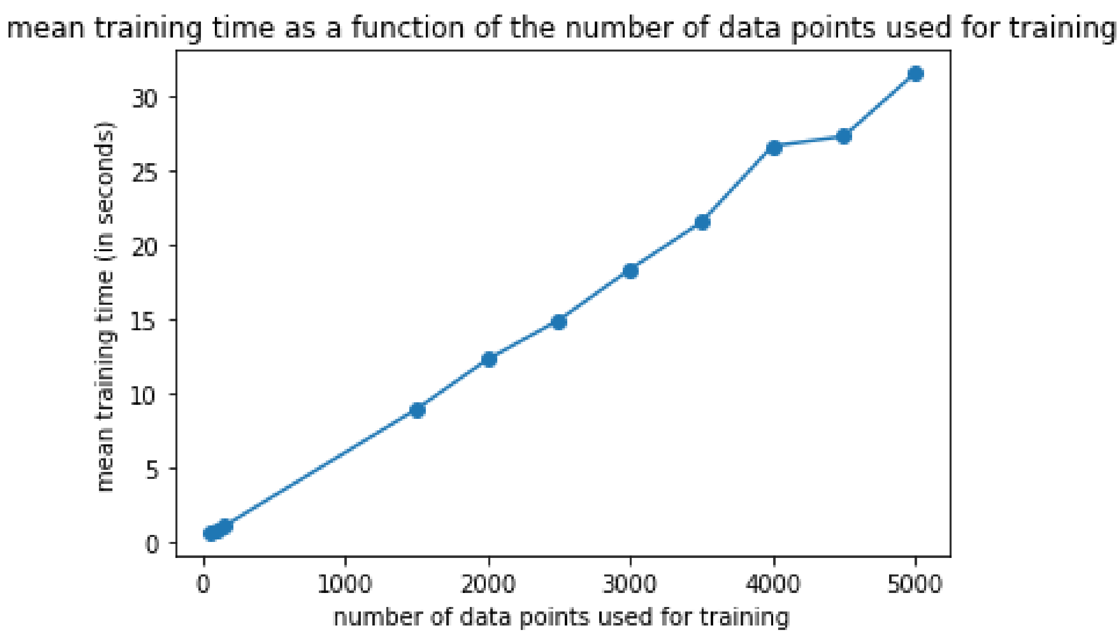

Figure 10: Digits dataset - training time as a function of the number of data points used for training

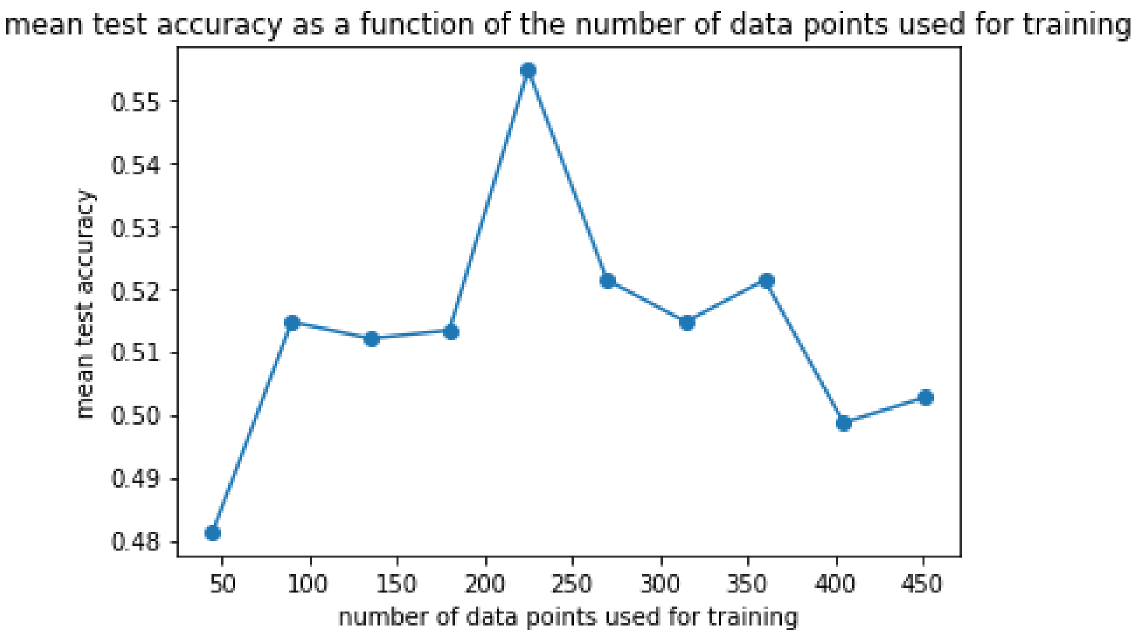

Figure 11: Faces dataset - mean test accuracy as a function of the number of data points used for training

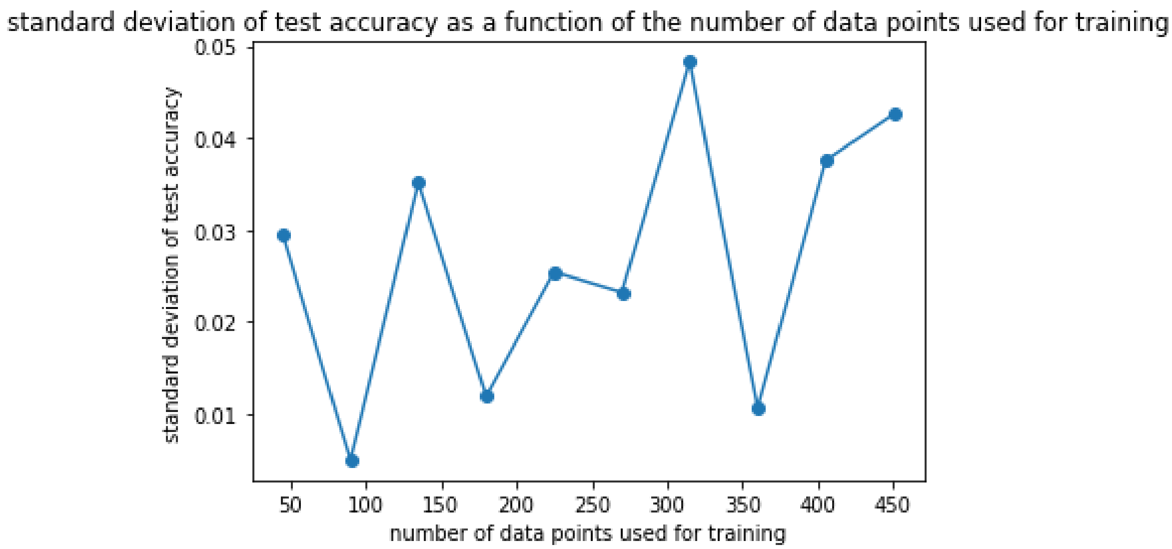

Figure 12: Faces dataset - standard deviation of test accuracy as a function of the number of data points used for training

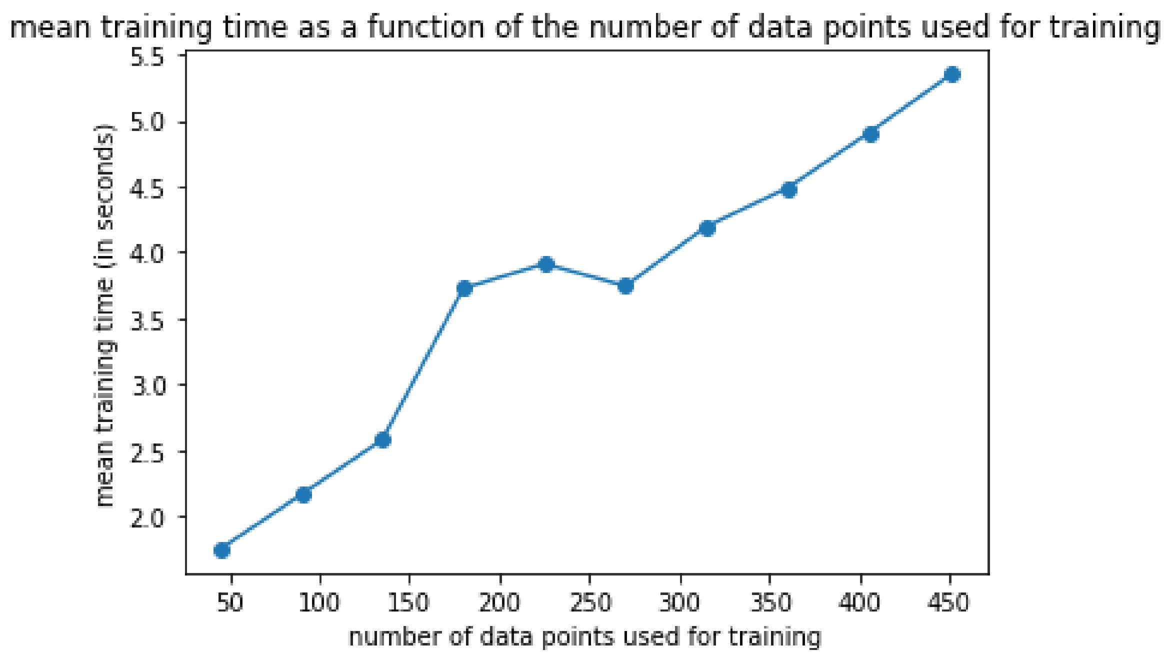

Figure 13: Faces dataset - training time as a function of the number of data points used for training

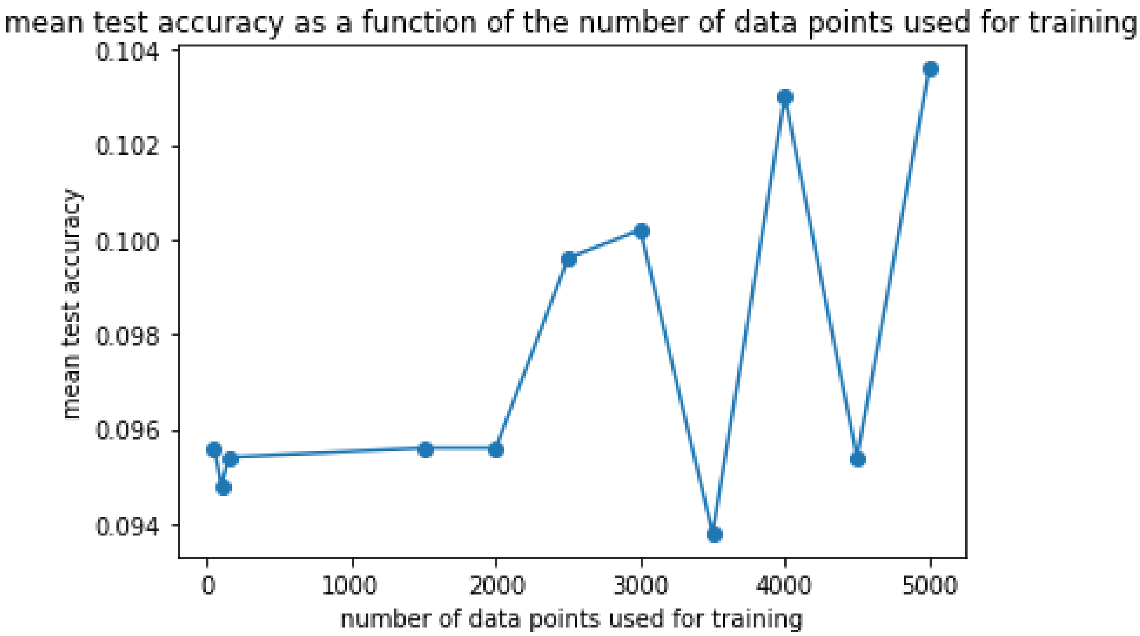

Figure 14: Digits dataset - mean test accuracy as a function of the number of data points used for training

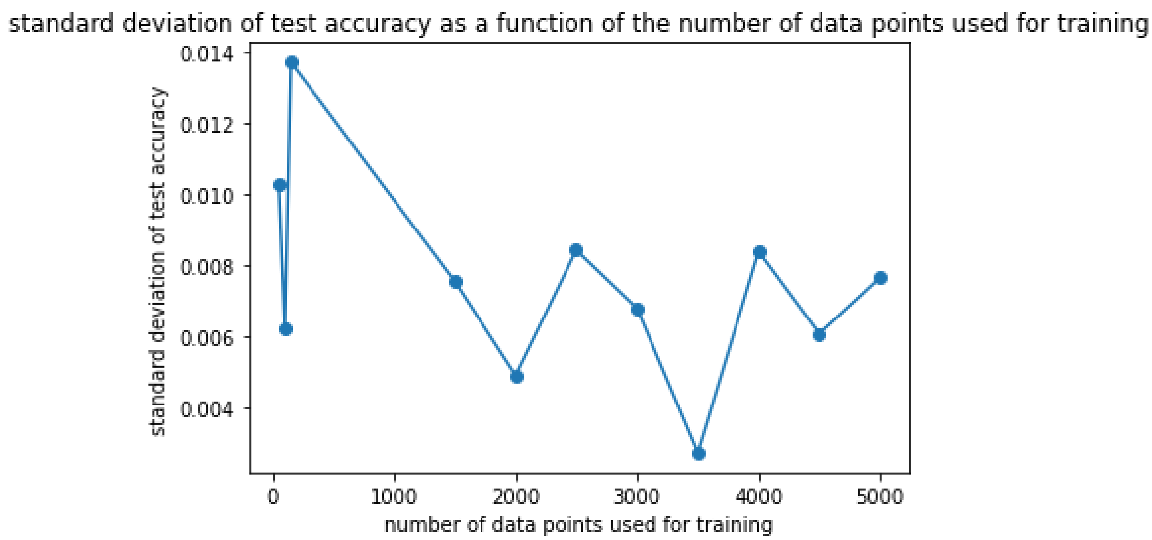

Figure 15: Digits dataset - standard deviation of test accuracy as a function of the number of data points used for training

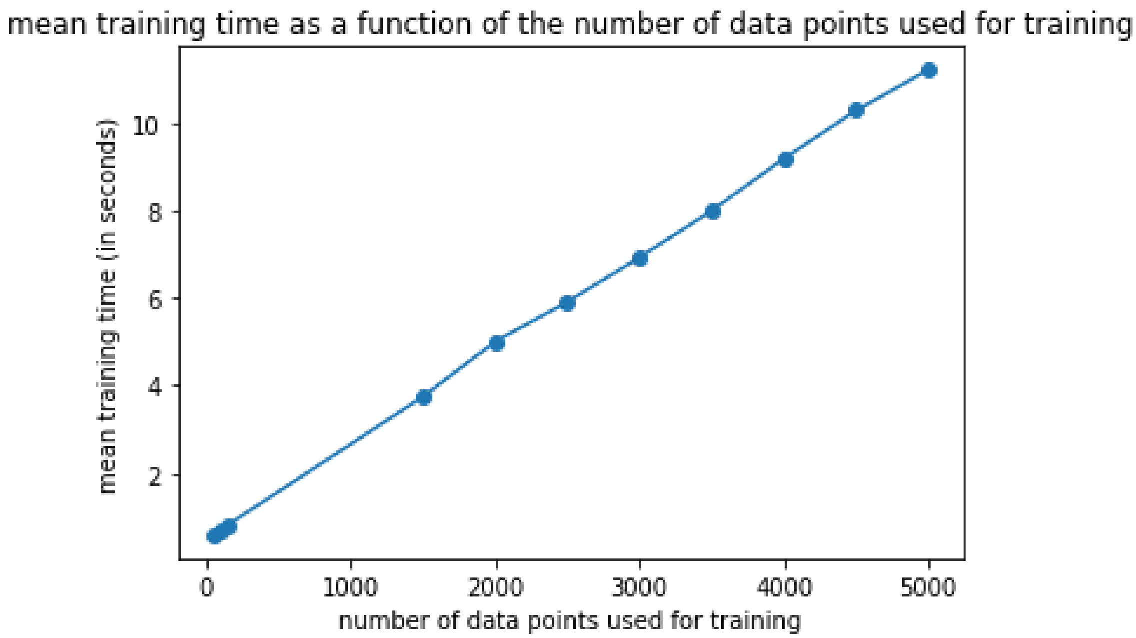

Figure 16: Digits dataset - training time as a function of the number of data points used for training