A.I. Midterm

Helen Ngo

October 12, 2016

1 Classification Task

Compare two algorithms on a classification task: the Pocket algorithm (designed for classification), and linear regression (not designed for classification). For linear regression, after learning the weights $\mathbf{w}$; we use $h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x})$ to classify $\mathbf{x}$. For the dataset, start by using the given code. Then create another dataset using the same methods as above, which we will use to estimate E_{out}. Try the following three approaches using multiple randomized experiments and explain which works best in terms of both E_{out} and the amount of computation required.

(a) The Pocket algorithm, starting from $\mathbf{w} = 0$.

(b) Linear regression (applied as a classification method).

(c) The Pocket algorithm, starting from the solution given by linear regression.

1.1 Classifying between 0 and Everything Else

When producing the dataset from "features.csv" where 0 is classified against everything else, we get the plot on the next page. Note that zero was assigned the positive class in blue. For this linearly inseparable dataset it took:

(a) The Pocket algorithm took 1223 iterations and approximately 4 minutes to converge, with $E_{out} \approx 0.1078$.

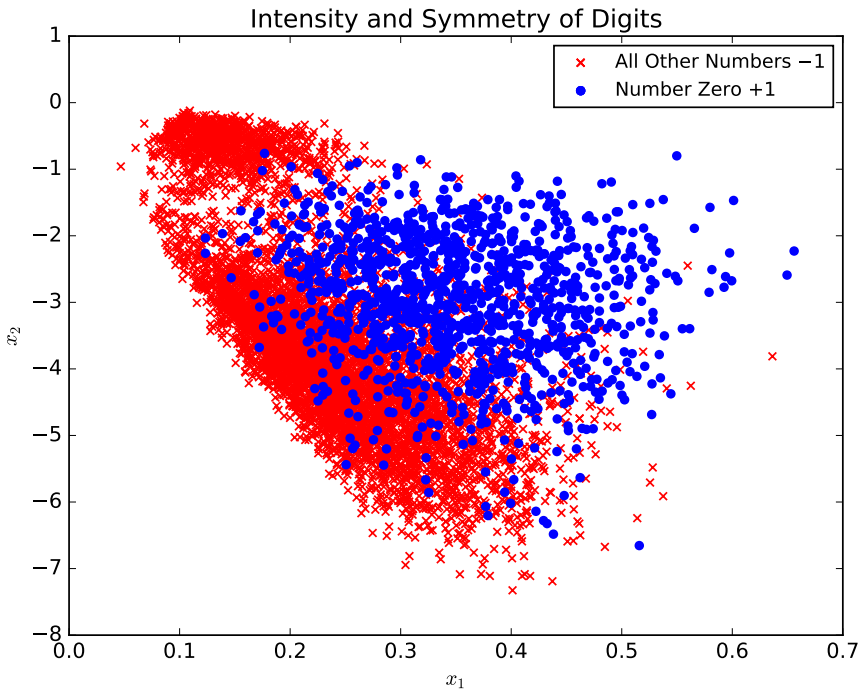
(b) Linear regression produced a hypothesis with $E_{out} \approx 0.1086$ in about 0.5 minutes.

(c) The Pocket algorithm starting from a linear regression solution took 864 iterations and 2.5 minutes to converge. The $E_{out} \approx 0.1059$.

It should be noted that convergence is either no classification error, more than 500 iterations with the same optimal hypothesis, or more than ten thousand total iterations, which can be seen in the pla function.

```
        # Iterate until all points are correctly classified
        while self.classification_error(w) != 0:
            it += 1
```

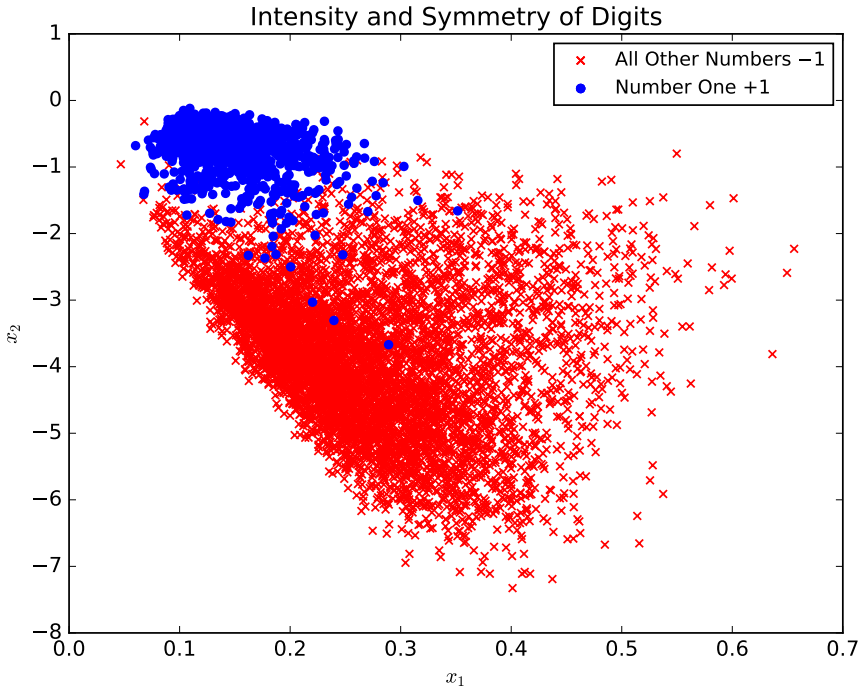
```
# Pick random misclassified point
x, y = self.choose_miscl_point(w0)
# Update weights
w0 += y*x
# Update if new weights are better
if self.classification_error(w)>self.classification_error(w0):
    w = copy.deepcopy(w0)
    count = 0
else:
    count += 1
# Converge after 500 iterations with the same wieghts
if count > 500:
    break
# Converge after 10000 iterations overall
if it > 10000:
    break
```



The Pocket algorithm starting from a linear regression, algorithm (c), had the lowest classification error at $E_{out} \approx 0.1059$, but it was only better than the hypothesis produced by linear regression, algorithm (b), which had a classification error of $E_{out} \approx 0.1086$. Algorithm (c) was less than 1% better at classifying the negative and positive classes but took approximately 5 times the amount of time as algorithm (b). The Pocket algorithm, algorithm (a), had the longest computing time and only classified 0.0010 more points better than linear regression.

1.2 Classifying between 1 and Everything Else

When producing the dataset from "features.csv" where 1 is classified against everything else, we get the plot below. Note that zero was assigned the positive class in blue.



For this linearly inseparable dataset it took:

(a) The Pocket algorithm took 1115 iterations and approximately 4 minutes to converge, with $E_{out} \approx 0.0121$.

(b) Linear regression produced a hypothesis with $E_{out} \approx 0.0152$ in about 0.5 minutes.

(c) The Pocket algorithm starting from a linear regression solution took 1151 iterations and 5 minutes to converge. The $E_{out} \approx 0.0121$.

For classifying between 1 and everything else, the algorithm (a) and algorithm (c) has the same classification error, $E_{out} \approx 0.0121$, but algorithm (c) took more iterations and about 1 more minute to converge. Once again, linear regression misclassified less than 1% more than the pocket algorithms, and only took about significantly less time than both algorithms (a) and (c).

1.3 Conclusion

The Pocket algorithm starting from a liner regression solution consistently produced the lowest E_{out} and could take from approximately the minimum number of iterations for convergence to the same amount as the normal Pocket algorithm. Therefore the number of computation is either similar to algorithm (a) or less, but the E_{out} will always be the lowest (possibly tied). Linear regression for these datasets produced a minusculy higher E_{out}, but at significantly less computing time. If you need to minimize the the E_{out}, the Pocket algorithm starting from a linear regression solution would be preferred, but for a computationally efficient overview, linear regression will probably do fine.

2 Problem 2.12

For an $\mathcal{H}$ with $d_{vc} = 10$, what sample size do you need (as prescribed by the generalized bound) to have a 95% confidence that your generalization error is at most 0.05?

2.1 Iterative Method

Using equation (2.13), we need

$$N \geq \frac{8}{0.05^2} \ln\left(\frac{4(2N)^{10} + 4}{0.05}\right),$$

where $N \in \mathbb{Z}$. We know that the equation converges as an iterative method, where each iteration

$$N_i \geq \frac{8}{0.05^2} \ln\left(\frac{4(2N_{i-1})^{10} + 4}{0.05}\right).$$

We know that the equation has converged when $N_i/N_{i-1} = 1$.

The python code for the equation as an iterative method:

```
import numpy as np
import copy


def  gen_error(N0=1000):
    while True:
        # Equation (2.13)
        N = (8/0.05**2)*np.log((4*(2*N0)**10+4)/0.05)
        # Make N an integer > N
        N = np.ceil(N)
        # Test for convergence
        if N/N0 == 1:
            break
        # Update with new guess
        N0 = copy.deepcopy(N)
    # Return sample size
    return N



def main():
    # Guess samle size of 1000
    sample_size = gen_error(N0=1000)
    print(sample_size)
main()
```

Running this python code we get $N = 452957$. Putting this into equation (2.13), we get

$$452957 \geq \frac{8}{0.05^2} \ln\left(\frac{4(2(452957))^{10}+4}{0.05}\right).$$

Thus $N = 452957$.

3 Drawing of the Professor