Linear Regression, Linear Classification, and Logistic Regression

Rachit Agarwal and Jay Mangrulkar

February 2019

Contents

1 Introduction

This project focuses on machine learning with an emphasis on three main topics:

- 1. Linear Regressions
- 2. Linear Classification
- 3. Logistic Regression

The following are the objectives of this assignment:

- 1. Implement linear regression using gradient descent.
- 2. Implement linear classifiers using the perception algorithm and logistic regression.
- 3. Experiment variations of the algorithms
- 4. Evaluate the classifier.
- 5. Present results and implementation in a dissertation.

These programs are built using Python3. In short, gradient descent is used to linear discriminant functions. First, this method is used to implement linear regression. Then, the perceptron algorithm is programmed and the threshold function is improved with the logistic curve.

2 Linear Regression

- The data sets used for this project are the letter counts in the first 15 chapters of the French and English version of *Salammbô*. In this data set, the first column corresponds to the total count of characters, while the second column is the count of A's.
- This part of the project was primarily provided to us. There is a function to compute the sum of squared errors (`sse()`) and one to normalize the data (`normalize()`).
- There are also two functions to apply stochastic and batch descent. These are: `stoch_descent()` and `batch_descent()`, respectively.
- The stochastic and batch descent are then applied and the coordinates for the data sets are restored and plotted on a line.
- The displayed graphs when the programs are run show:
  1. The plotted coordinates.
  2. The errors
  3. The weights for each descent and their updates

3 Perceptron

- This part uses the same data set as part 1 of the assignment. The same methods handle both linear classification and logistic regression.
- The scaling function `scale_data()` scales/normalizes a data array such all values `n` are between 0 and 1.
- The reader function `read_libsvm_and_scale()` reads the data file formatted in LIBsvm and returns both unscaled and scaled versions (as a tuple) of a dictionary mapping (x,y) tuples to language, where 0 = French and 1 = English.
- The prediction function `threshold()` maps some input $\mathbf{x}$ (a real-valued vector) to a predicted output value $\mathbf{f(x)}$.

  - Linear Classification: Returns 1 if the dot product of the input and weight vectors is positive, and returns 0 if it is negative.
  - Logistic Regression: Returns the probability of an output value of 1 (English) using the stochastic logistic regression formula, given below (w and x are weight and input vectors, respectively):

$$\frac{1}{1 + e^{-w \cdot x}}$$

- The misclassified count function `get_num_misclassified()` that counts the number of misclassified examples given weight and input vectors.
- The training function `train()` trains the classifier by iterating over all elements in the data set and updating the weight vector accordingly, until the number of misclassified items is below a given threshold. This is repeated for the given number of trials.
- The evaluation function `leave_one_out_cross_validation()` performs leave-one-out cross validation (a special case of k-fold cross validation in which k = n) on a data set. Essentially, it loops through each data point and predicts the output for that data point using the rest of the data as the data set (i.e. removes that point from the data set).

4 Implementation

To run the program, navigate to the project directory at `/h/d5/x/ra3160ag-s/edaf70-p3-perceptron`. The code and accompanying comments and results/graphs are contained within a Jupyter Notebook entitled `main.ipynb`. To run the notebook, run `jupyter notebook` and select `main.ipynb` in the notebook viewer. Each code block can then be run with `SHIFT + ENTER`.

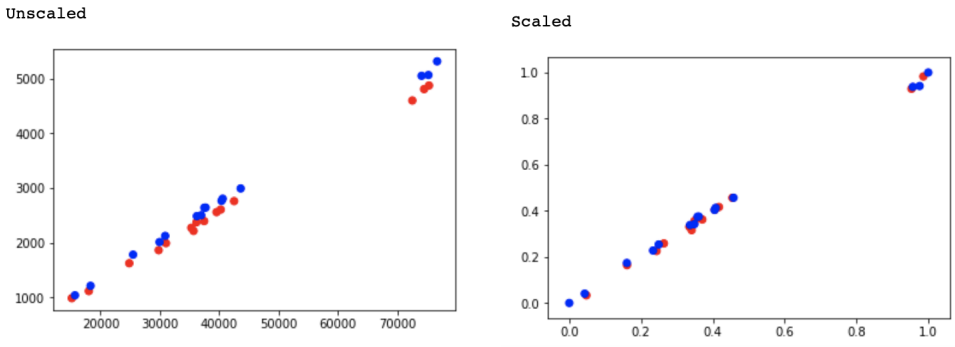
The code is also viewable in the GitHub repository `rachitag22/edaf70-p3-perceptron`, at `https://github.com/rachitag22/edaf70-p3-perceptron`.

5 Results & Discussion

Below is a snapshot of the example data provided to us. For each language, the first column is the total count of characters in each chapter, and the second column is the total count of letter A's in each chapter.

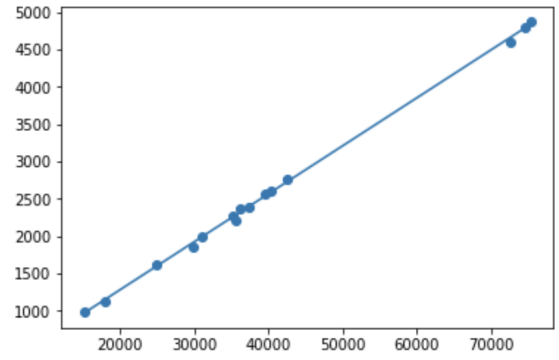
	English		French	
Chapter	# Chars	# A's	# Chars	# A's
1	35680	2217	36961	2503
2	42514	2761	43621	2992
3	15162	990	15694	1042
4	35298	2274	36231	2487
5	29800	1865	29945	2014
6	40255	2606	40588	2805
7	74532	4805	75255	5062
8	37464	2396	37709	2643
9	31030	1993	30899	2126
10	24843	1627	25486	1784
11	36172	2375	37497	2641
12	39552	2560	40398	2766
13	72545	4597	74105	5047
14	75352	4871	76725	5312
15	18031	1119	18317	1215

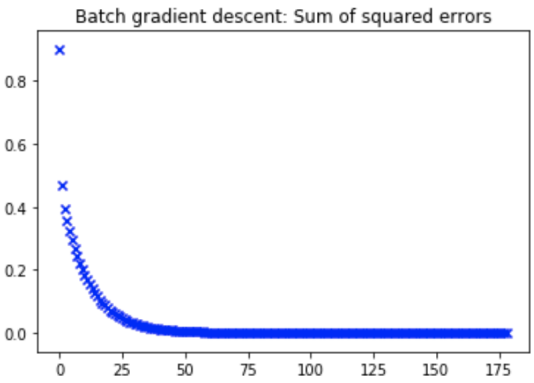
A cursory plot at the data is useful in providing a visual indicator of the data's linear separability. Below, English data points are in red, and French in blue.

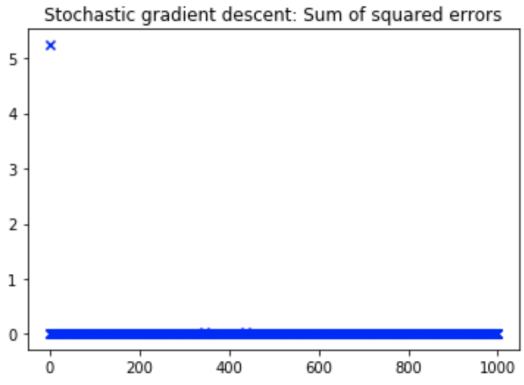


5.1 Linear Regression

Although we used our teacher's code for linear regression, it's definitely worthwhile looking at the outputted graphs. There are many, but here are some rather significant ones. We see that the regression line matches the data points fairly well. For batch descent, the sum of squared errors decreases in a manner similar to `-ln x`, i.e. very fast then slowly approaches zero. For stochastic descent, it drops to zero after just one iteration/epoch.







For linear classification and logistic regression, we performed both our own tests and leave-one-out cross validation, on both the unscaled data and scaled (normalized) data.

Our own test was composed of running `n` trials and seeing how many iterations it would take for the number of misclassified examples to dip below a given threshold.

Our leave-one-out cross validation test performed k-fold cross validation with k = n, explained in detail above in the method description.

5.2 Linear Classification

Basic Tests

For the unscaled data (both English and French data), we ran 100 trials and set the threshold to 0 (meaning each trial would run until every example was classified correctly). The weight vector with the lowest number of iterations (8) to become "perfect" was `<-617, 9380>`.

However, when we ran the same tests (100 trials) on the scaled data (again, both languages), the infinite loop limit would always be reached for most thresholds up to 12 or 13. This means that even after the limit (of 1000 iterations) was reached, the minimum miss count would never dip below the threshold. We noticed that the minimum number of iterations jumped from 1 to ¿1000 at around threshold 13. This was fairly odd to us, as the classifier was monumentally better when looking at non-normalized data. The best-performing weight vector was `<-0.27, -0.27>` which is extremely smaller in magnitude than unscaled set's best vector.

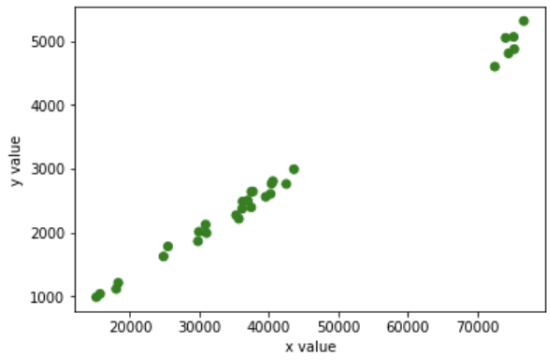
```
Unscaled Data

Miss Threshold: 0
Total # trials: 100
Mean # iterations: 94
Median # iterations: 88
Max # iterations: 269

Best (Min #) iterations: 8
Best Weight Vector: [-618.57134097 9379.65093025]
Best Num Misses : 0
```

```
Scaled Data

Miss Threshold: 12
Total # trials: 100
Mean # iterations: 1000
Median # iterations: 1000
Max # iterations: 1000

Best (Min #) iterations: 1000
Best Weight Vector: [-1.29438717 -0.85195291]
Best Num Misses : 14

Scaled Data

Miss Threshold: 14
Total # trials: 100
Mean # iterations: 1
Median # iterations: 1
Max # iterations: 1

Best (Min #) iterations: 1
Best Weight Vector: [-0.26633162 -0.26185262]
Best Num Misses : 13
```

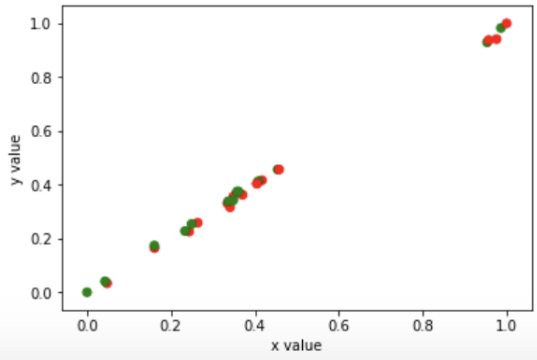
Leave-One-Out Cross Validation

Similar to our basic trial-based tests, the classifier was great with scaled data when we performed leave-one-out cross validation. We ran 10 trials for each data point (both languages) and used the best-performing weight vector. While the classifier had perfect accuracy with the unscaled set, it was much worse with the scaled data (46% accuracy). Below, zero-error tuples are marked in green, and error-of-one tuples are marked in red.

```
Unscaled Data
Average Error: 0
# Correct Predictions (error = 0, green): 30
# Incorrect Predictions (0 < error < 0.5, yellow): 0
# Very Incorrect Predictions (0.5 < error <= 1, red): 0
Accuracy Rate: 100.0%
```




```
Scaled Data
Average Error: 0.5357142857142857
# Correct Predictions (error = 0, green): 13
# Incorrect Predictions (0 < error < 0.5, yellow): 0
# Very Incorrect Predictions (0.5 < error <= 1, red): 15
Accuracy Rate: 46.42857142857143%
```



5.3 Logistic Regression

Basic Tests

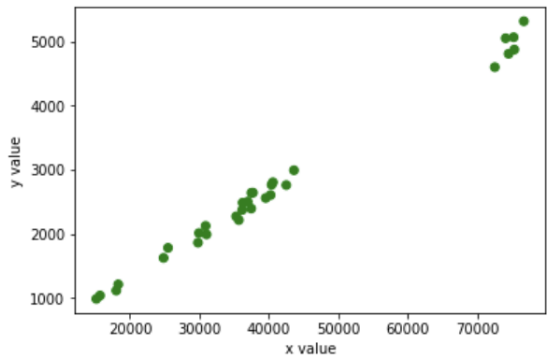
In our basic testing for the logistic regression classifier, we saw very similar results to that of the linear one. The unscaled data had a best iteration count of 10 to reach "perfection," with a weight vector of much higher magnitude than the scaled data's. For the scaled data, we saw the same "jump" in minimum iterations at around the same threshold value.

```
Unscaled Data

Miss Threshold: 0
Total # trials: 100
Mean # iterations: 92
Median # iterations: 92
Max # iterations: 247

Best (Min #) iterations: 15
Best Weight Vector: [-1329.0311246  20235.82249613]
Best Num Misses : 0

Scaled Data

Miss Threshold: 12
Total # trials: 100
Mean # iterations: 1000
Median # iterations: 1000
Max # iterations: 1000

Best (Min #) iterations: 1000
Best Weight Vector: [-12.92994459  13.17074247]
Best Num Misses : 13

Scaled Data

Miss Threshold: 14
Total # trials: 100
Mean # iterations: 1
Median # iterations: 1
Max # iterations: 1

Best (Min #) iterations: 1
Best Weight Vector: [0.10193095 0.14726132]
Best Num Misses : 13
```

Leave-One-Out Cross Validation

Just like we saw in the linear classifier's k-fold testing, the logistic classifier was perfect with unscaled data but performed worse with scaled data. However, accuracy rate for the scaled data here was 0% as opposed to the 46% from the linear classifier. Since the threshold function returns a prediction between 0 and 1, graph coloring is a bit different here. Green means an error of 0, yellow means an error between 0 and 0.5, and red means an error between 0.5 and 1 (inclusive).

```
Unscaled Data
Average Error: 0.0
# Correct Predictions (error = 0, green): 30
# Incorrect Predictions (0 < error < 0.5, yellow): 0
# Very Incorrect Predictions (0.5 < error <= 1, red): 0
Accuracy Rate: 100.0%
```




```
Scaled Data
Average Error: 0.5023820215865707
# Correct Predictions (error = 0, green): 0
# Incorrect Predictions (0 < error < 0.5, yellow): 15
# Very Incorrect Predictions (0.5 < error <= 1, red): 13
Accuracy Rate: 0.0%
```