CSC 566 Assignment 3 - Linear Classifiers

Aidan Barbieux
Cal Poly San Luis Obispo
`abarbieu@calpoly.edu`

February 5, 2022

Contents

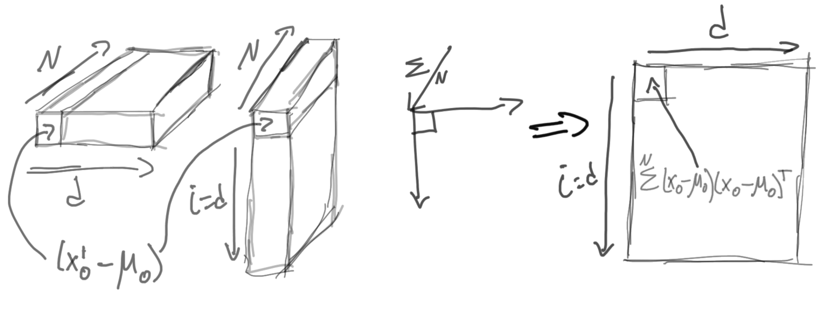
List of Tables and Figures

1 Implementation

I chose to implement gradient descent for both methods, and found convergence to not be an issue for the Iris dataset. As such, I used a relatively large η of 0.1 for both. Convergence was assessed by the magnitude of the error derivative magnitude, the cutoff was decided by running 200 iterations of gradient descent, and seeing at what magnitude value convergence settled at. This was determined to be 0.1 and 0.01 for LDA and logistic regression, respectively.

To simplify some calculations (and readability/understanding of underlying math) I utilized Einstein summation notation implemented by `np.einsum`. This allowed me to calculate S_0 is one linear algebra operation that took the "left" product of $X_0 - \mu_0$ and itself, represented in the form: $(X_0 - \mu_0)_d^N (X_0 - \mu_0)_i^N = C_i^d$ or $np.einsum(\text{"}Nd, Ni \rightarrow id\text{"}, X_0 - \mu_0, X_0 - \mu_0)$. Geometrically, this looks like:

Figure 1: Geometric representation of Einstein summation notation to calculate S_0



I found this to increase performance by some tangible margin when repeatedly running LDA, and simplified my implementation.

The final detail of implementation was in regards to the intercept of w for logistic regression. I decided to simply add a column of 1's to the data passed in, and added an additional dimension to w to account for it. When trying to do this for LDA, gradient descent would leave the intercept at the original random value, as the scatter and centroid distance were both 0 as every datapoint occupied the exact same position. Thankfully, it doesn't matter "where" w lies in space for LDA, and thus I ignored the intercept variable, so all w's would be unit vectors passing through the origin.

2 Results

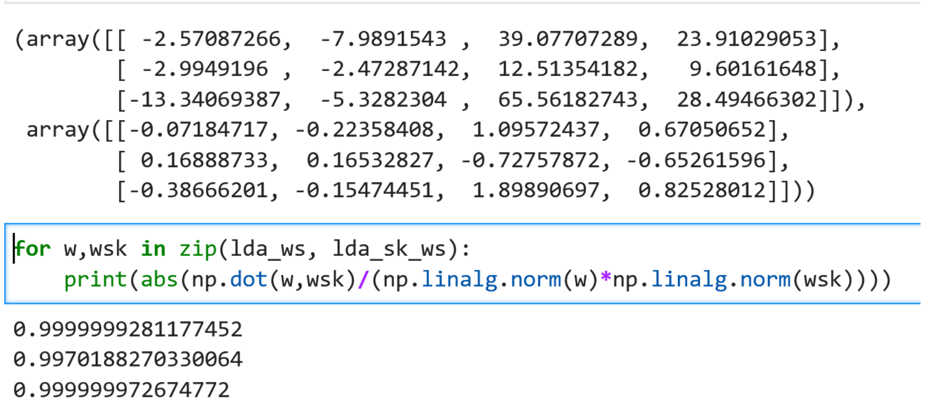
2.1 Settings

All runs were conducted with an η of 0.1 and gradient descent stopped when the magnitude of the gradient was below 0.1 and 0.01 for LDA and logistic regression, respectively.

2.2 Model Comparisons

I compared models by taking the cosine similarity of the resulting w's, shown below:

Figure 2: Cosine similarities between my models and sklearn's for LDA

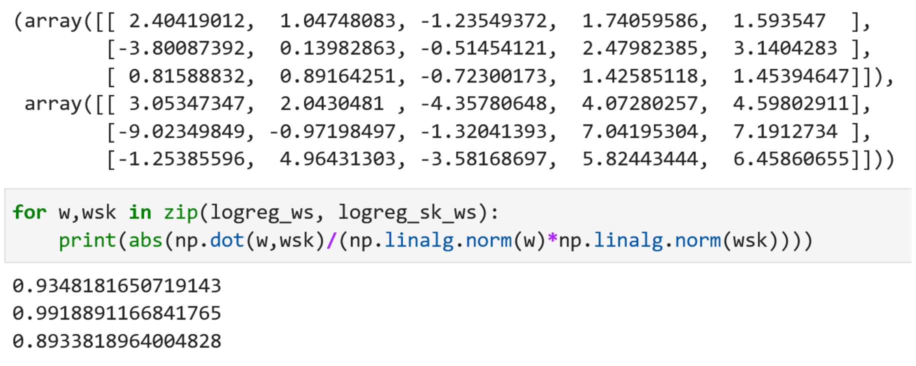


```
(array([[ -2.57087266,  -7.9891543 ,  39.07707289,  23.91029053],
        [ -2.9949196 ,  -2.47287142,  12.51354182,   9.60161648],
        [-13.34069387,  -5.3282304 ,  65.56182743,  28.49466302]]),
 array([[-0.07184717, -0.22358408,  1.09572437,  0.67050652],
        [ 0.16888733,  0.16532827, -0.72757872, -0.65261596],
        [-0.38666201, -0.15474451,  1.89890697,  0.82528012]]))
```

```
for w,wsk in zip(lda_ws, lda_sk_ws):
    print(abs(np.dot(w,wsk)/(np.linalg.norm(w)*np.linalg.norm(wsk))))
```

```
0.9999999281177452
0.9970188270330064
0.999999972674772
```

Figure 3: Cosine similarities between my models and sklearn's for logistic regression



```
(array([[ 2.40419012,  1.04748083, -1.23549372,  1.74059586,  1.593547  ],
        [-3.80087392,  0.13982863, -0.51454121,  2.47982385,  3.1404283 ],
        [ 0.81588832,  0.89164251, -0.72300173,  1.42585118,  1.45394647]]),
 array([[ 3.05347347,  2.0430481 , -4.35780648,  4.07280257,  4.59802911],
        [-9.02349849, -0.97198497, -1.32041393,  7.04195304,  7.1912734 ],
        [-1.25385596,  4.96431303, -3.58168697,  5.82443444,  6.45860655]]))
```

```
for w,wsk in zip(logreg_ws, logreg_sk_ws):
    print(abs(np.dot(w,wsk)/(np.linalg.norm(w)*np.linalg.norm(wsk))))
```

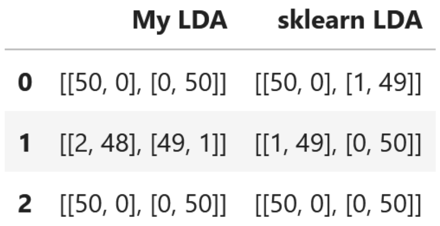
```
0.9348181650719143
0.9918891166841765
0.8933818964004828
```

With different starting w's, the results would shift slightly, but overall they remained very similar. Interestingly, when I only used S_0 in place of S for LDA, the similarity for irisB (the middle similarity) went down, but my prediction accuracy beat sklearn's, with few other losses. When $S = S_0 + S_1$ was properly, the models matched but my predictions were still more accurate (seen in confusion matrices below). The matrices are in the form:

$$[Actual0 : [Predicted0, Predicted1], [Actual1 : [Predicted0, Predicted1]]$$

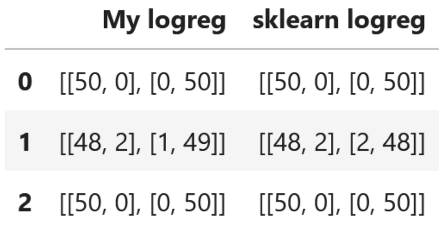
(if they are perfectly reversed, the comparison between above/below the line was flipped, and all class 0's were predicted as class 1's, and vice versa, meaning results must be read backwards)

Figure 4: Confusion matrices for LDA



	My LDA	sklearn LDA
0	[[50, 0], [0, 50]]	[[50, 0], [1, 49]]
1	[[2, 48], [49, 1]]	[[1, 49], [0, 50]]
2	[[50, 0], [0, 50]]	[[50, 0], [0, 50]]

Figure 5: Confusion matrices for logistic regression



	My logreg	sklearn logreg
0	[[50, 0], [0, 50]]	[[50, 0], [0, 50]]
1	[[48, 2], [1, 49]]	[[48, 2], [2, 48]]
2	[[50, 0], [0, 50]]	[[50, 0], [0, 50]]

As you can see, they are all very similar except for LDA on irisBX (1). Here, my model produced accurate results, but sklearn's guessed everything to be of class 1. I'm unsure why sklearn's predictions are so poor, as the model is the same, and separation is visually apparent, and similar to my separation:

3 Model Choice

It seems like logistic regression is the more powerful predictor for the iris dataset, but both do very well. I would lean towards my LDA over sklearn's using the lsqe solver, but perhaps with a different solver sklearn's LDA would be better.

Figure 6: Separation of classes from my LDA

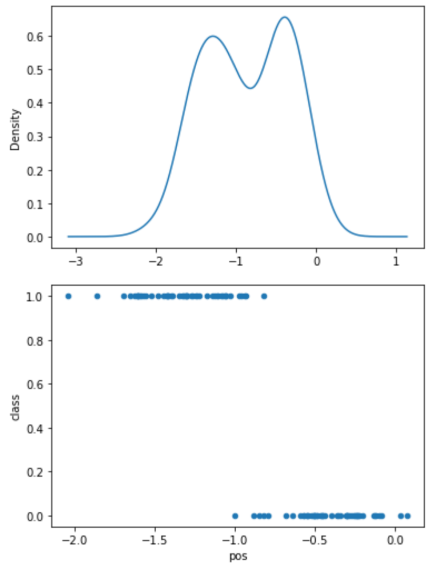
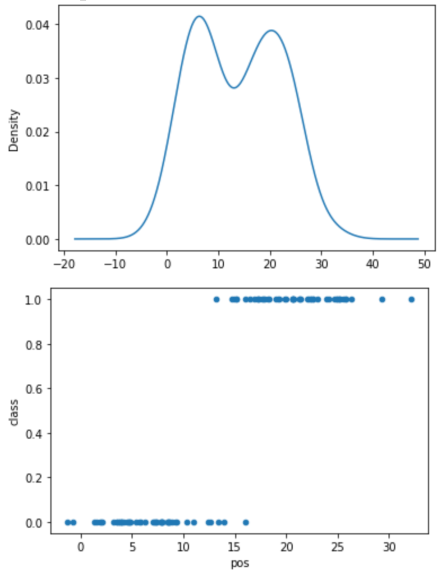


Figure 7: Separation of classes from sklearn's LDA


Density
class
pos