hw7

December 2, 2024

1 Homework 7 - Ishaan Sathaye

1.1 Part A: Theory

In the following, you might find the following notation useful:

$1_{A,i1} = 1$ if observation i is in class A

$1_{A,i1} = 0$ if observation i is not in class A

Consider this Neural Network, in which we have: - Three sets of weights and biases for the initial layer, $\overrightarrow{w_1}$, $\overrightarrow{w_2}$, $\overrightarrow{w_3}$; c_1, c_2, c_3 - One weight set and bias for the hidden layer, $\vec{v} = (v_1, v_2, v_3)$; d - A sigmoid activation function, $g(u) = \frac{1}{1+e^{-u}}$ - A loss function of squared-error on the predicted probabilities of Class 1, compared to y values of 0 or 1:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\hat{p_i} - y_i)^2$$

1.1.1 Question 1 Second Layer Gradients

Give the gradient equation for v_1, v_2, v_3, and d. (When possible feel free to say "the rest is the same as for the other one" rather than repeating equations)

- Gradient for v_j is the same as for v_1 for $j = 2, 3$
  - $u = v_1 g_1 + v_2 g_2 + v_3 g_3 + d$
  - $g_j = g(w_j^T x_i + c_j)$
  - $\frac{\partial L}{\partial v_j} = \frac{2}{n}\sum_{i=1}^{n}(\hat{p_i} - y_i)(\hat{p_i})(1 - \hat{p_i})g_j$
- Gradient for d:
  - $\frac{\partial L}{\partial d} = \frac{2}{n}\sum_{i=1}^{n}(\hat{p_i} - y_i)(\hat{p_i})(1 - \hat{p_i})$

1.1.2 Question 2 First Layer Gradients

Give the gradient equations for $w_{1,1}$, $w_{1,2}$, ..., $w_{3,p}$, c_1, c_2, c_3.

- Gradient for $w_{j,k}$:
  - $\frac{\partial L}{\partial w_{j,k}} = \frac{2}{n}\sum_{i=1}^{n}(\hat{p_i} - y_i)(\hat{p_i})(1 - \hat{p_i})v_j g_j'(w_j^T x_i + c_j)x_{i,k}$
    - the term $g_j'(w_j^T x_i + c_j)$ is equal to \$ g(w^T_jx_i + c_j)(1 - g(w^T_jx_i + c_j))\$
- Gradient for c_j:
  - $\frac{\partial L}{\partial c_j} = \frac{2}{n}\sum_{i=1}^{n}(\hat{p_i} - y_i)(\hat{p_i})(1 - \hat{p_i})v_j g_j'(w_j^T x_i + c_j)$
  - the rest is the same as for the other ones

1.2 Part B: Computer Implementation

[147]:
```
import pandas as pd

cannabis = pd.read_csv('../hw3/cannabis_full.csv')
cannabis_hybrid = cannabis[cannabis['Type'] == 'hybrid']

# remove hybrid strains
cannabis = cannabis[cannabis['Type'] != 'hybrid']

# remove effects and flavors
cannabis = cannabis.drop(columns=['Effects', 'Flavor'])

# map type to 0 or 1
cannabis['Type'] = cannabis['Type'].map({'indica': 0, 'sativa': 1})

# drop null values
cannabis = cannabis.dropna()

cannabis.head()
```

[147]:
```
       Strain  Type  Rating  Creative  Energetic  Tingly  Euphoric  Relaxed  \
2        1024     1     4.4       1.0        1.0     0.0       0.0      1.0
5  3-Bears-Og     0     0.0       0.0        0.0     0.0       0.0      0.0
7      303-Og     0     4.2       0.0        0.0     0.0       1.0      1.0
8      3D-Cbd     1     4.6       0.0        0.0     0.0       0.0      1.0
9    3X-Crazy     0     4.4       0.0        0.0     1.0       1.0      1.0

   Aroused  Happy  …  Ammonia  Minty  Tree  Fruit  Butter  Pineapple  Tar  \
2      0.0    1.0  …      0.0    0.0   0.0    0.0     0.0        0.0  0.0
5      0.0    0.0  …      0.0    0.0   0.0    0.0     0.0        0.0  0.0
7      0.0    1.0  …      0.0    0.0   0.0    0.0     0.0        0.0  0.0
8      0.0    1.0  …      0.0    0.0   0.0    0.0     0.0        0.0  0.0
9      0.0    1.0  …      0.0    0.0   0.0    0.0     0.0        0.0  0.0

   Rose  Plum  Pear
2   0.0   0.0   0.0
5   0.0   0.0   0.0
7   0.0   0.0   0.0
8   0.0   0.0   0.0
9   0.0   0.0   0.0

[5 rows x 67 columns]
```

[148]:
```
from sklearn.linear_model import LogisticRegression
```

```
effect_cols = cannabis.columns[cannabis.columns.get_loc("Creative"):cannabis.
  ↪columns.get_loc("Mouth")+1]
flavors_cols = cannabis.columns[cannabis.columns.get_loc("Earthy"):cannabis.
  ↪columns.get_loc("Pear")+1]

# Logistic Regression using only effect predictors
m1 = LogisticRegression()
m1.fit(cannabis[effect_cols], cannabis['Type'])

# Logistic Regression using only flavor predictors
m2 = LogisticRegression()
m2.fit(cannabis[flavors_cols], cannabis['Type'])

# Logistic Regression using only rating predictor
m3 = LogisticRegression()
m3.fit(cannabis[['Rating']], cannabis['Type'])
```

[148]:
```
LogisticRegression()
```

[149]:
```
import numpy as np

w1 = np.concatenate([m1.coef_[0], np.zeros(len(flavors_cols) + 1)])
w2 = np.concatenate([np.zeros(len(effect_cols)), m2.coef_[0], np.zeros(1)])
w3 = np.concatenate([np.zeros(len(effect_cols) + len(flavors_cols)), m3.
  ↪coef_[0]])

c1 = m1.intercept_[0]
c2 = m2.intercept_[0]
c3 = m3.intercept_[0]
```

1.2.1 Question 1: Optimizing the final layer

Update your code from Homework 6 so that the final layer parameters are chosen via gradient descent. (Keep the “cheater” w’s you that you implemented last week.)

[150]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sigmoid_derivative(z):
    sig = sigmoid(z)
    return sig * (1 - sig)

# stopping condition
def check_stopping_condition(v, d, threshold=1e-4):
    return np.linalg.norm(v) < threshold and np.abs(d) < threshold

def loss(output, target):
```

```python
    return (output - target) ** 2

def gd_perceptron(train_pred, train_target, w1, w2, w3, c1, c2, c3,␣
 ↪learning_rate=0.01, epochs=1000):
    # initialize v and d
    v = np.array([1/3, 1/3, 1/3])
    d = 0
    for epoch in range(1, epochs+1):
        total_loss = 0
        for preds, target in zip(train_pred, train_target):
            preds = np.array(preds)
            z1 = np.dot(preds, w1) + c1
            z2 = np.dot(preds, w2) + c2
            z3 = np.dot(preds, w3) + c3
            u1 = sigmoid(z1)
            u2 = sigmoid(z2)
            u3 = sigmoid(z3)

            u = np.array([u1, u2, u3])

            # hidden layer
            hidden_layer_input = np.dot(u, v) + d
            output = sigmoid(hidden_layer_input)

            # loss
            # pred_type = 1 if output >= 0.5 else 0
            current_loss = loss(output, target)
            total_loss += current_loss

            # backpropagation
            output_error = output - target
            deriv = sigmoid_derivative(output)

            grad_v = output_error * deriv * u
            grad_d = output_error * deriv

            if check_stopping_condition(grad_v, grad_d, 1e-4):
                break

            v -= learning_rate * grad_v
            d -= learning_rate * grad_d

    return v, d
```

1.2.2 Question 2: Best final layer

Fit your function on the same cannabis data.

```
[151]: cannabis_preds = cannabis[effect_cols.tolist() + flavors_cols.tolist() +␣
        ↪['Rating']].values
       true_labels = cannabis['Type'].values

       v, d = gd_perceptron(cannabis_preds, true_labels, w1, w2, w3, c1, c2, c3)
       v,d
```

```
[151]: (array([ 6.17362546,  4.19162658, -1.16314   ]), -4.340883764684458)
```

1.2.3 Question 3: Optimizing everything

Now update your code so that all parameters are chosen via gradient descent and back propagation.

```
[152]: def check_stopping_condition_all(v, d, w1, w2, w3, c1, c2, c3, threshold=1e-4):
           return (np.linalg.norm(v) < threshold and np.abs(d) < threshold and
           np.linalg.norm(w1) < threshold and np.linalg.norm(w2) < threshold and
           np.linalg.norm(w3) < threshold and np.abs(c1) < threshold and
           np.abs(c2) < threshold and np.abs(c3) < threshold)

       def gd_perceptron_all(train_pred, train_target, w1, w2, w3, c1, c2, c3,␣
        ↪learning_rate=0.01, epochs=500):
           # initialize v and d
           v = np.array([1/3, 1/3, 1/3])
           d = 0

           for epoch in range(1, epochs+1):

               total_loss = 0

               for preds, target in zip(train_pred, train_target):
                   preds = np.array(preds)

                   # forward pass
                   z1 = np.dot(preds, w1) + c1
                   z2 = np.dot(preds, w2) + c2
                   z3 = np.dot(preds, w3) + c3
                   u1 = sigmoid(z1)
                   u2 = sigmoid(z2)
                   u3 = sigmoid(z3)
                   u = np.array([u1, u2, u3])

                   # hidden layer
                   hidden_layer_input = np.dot(u, v) + d

                   output = sigmoid(hidden_layer_input)

                   # loss
```

```
            current_loss = loss(output, target)
            total_loss += current_loss

            # backpropagation
            output_error = output - target
            deriv = sigmoid_derivative(output)

            # gradients for v and d
            grad_v = output_error * deriv * u
            grad_d = output_error * deriv

            grad_u = output_error * deriv * v
            grad_z1 = grad_u[0] * sigmoid_derivative(z1)
            grad_z2 = grad_u[1] * sigmoid_derivative(z2)
            grad_z3 = grad_u[2] * sigmoid_derivative(z3)
            grad_w1 = grad_z1 * preds
            grad_w2 = grad_z2 * preds
            grad_w3 = grad_z3 * preds
            grad_c1 = grad_z1
            grad_c2 = grad_z2
            grad_c3 = grad_z3

            w1 -= learning_rate * grad_w1
            w2 -= learning_rate * grad_w2
            w3 -= learning_rate * grad_w3
            c1 -= learning_rate * grad_c1
            c2 -= learning_rate * grad_c2
            c3 -= learning_rate * grad_c3
            v -= learning_rate * grad_v
            d -= learning_rate * grad_d

        if check_stopping_condition_all(v, d, w1, w2, w3, c1, c2, c3, 1e-4):
            break

    return w1, w2, w3, c1, c2, c3, v, d
```

1.2.4 Question 4 Fit

Fit your Neural Network on the cannabis data.

(You will not lose points on this problem if your function does not successfully converge; however, I need to see that it is successfully running at least a few backprop steps.)

[153]:
```
w1, w2, w3, c1, c2, c3, v, d = gd_perceptron_all(cannabis_preds, true_labels,␣
 ↪w1, w2, w3, c1, c2, c3)

print("w1:", w1)
print("w2:", w2)
print("w3:", w3)
print("c1:", c1)
print("c2:", c2)
print("c3:", c3)
print("v:", v)
print("d:", d)
```

```
w1: [-0.71683973  2.98361483 -1.13090688 -0.25081201 -1.3406478   0.17277603
  0.94300489 -0.3105249   0.57014961  1.30562695 -1.18385025  0.67749114
 -2.63490791  0.          0.          0.16610538  0.54686096  1.43002331
  0.06842544 -0.06504742  0.59203785 -0.85776847  0.47336835 -0.91235034
  0.11868051 -0.37160788 -0.18106447  0.28056321 -1.23754765  0.57814552
 -0.09365098  0.00646762  0.30968857  0.31165338 -0.82152614  1.08374577
  0.21346203  0.06897588  0.07067305  0.46164007  0.16819764 -0.77970105
 -0.4219564  -0.07757994  0.20624473  0.08880509 -0.338095   -1.45516466
 -0.59016466  0.0114926   0.04636659  0.11367954  0.07131531  0.78121495
 -0.26777455 -0.34529515 -0.29139766 -0.29139766 -0.47090873  1.04674478
  0.8892645  -0.0827357   0.50538801 -0.02442583  0.11189354]
w2: [ 1.06450855 -0.03103494  0.81646611 -0.2049515  -0.9872233  -0.67497826
 -0.25353403  1.38377553 -1.02675742  0.07324382  0.00399838  1.31672537
 -0.40019272  0.          0.         -0.00383132  0.11247696 -0.11790633
 -0.95310943  0.42097865  0.92664024  0.91383766  0.1848128   0.38926831
  1.63199994  0.72010163  1.15621815 -1.18879971 -1.14508147 -0.30378735
  0.19416938 -0.92457994 -0.79891725 -0.18395466  0.74682932 -0.60275298
  0.33755323  0.70375702  0.78142495  0.39727889  0.21367641 -0.90707047
 -1.26535766  0.70725135 -0.14326176  1.0325022   1.06890509 -0.00977085
 -0.47314717  0.1269541  -1.57592055 -0.66896753  0.19712719 -0.42090498
 -0.9255905  -0.0467475   0.42286666  0.42286666 -1.84229668  1.96668933
  1.18781973 -0.84969528  0.14509415 -0.39076832  0.09553247]
w3: [-0.00316988 -0.35704534  0.20277953 -0.02026589  0.1086787  -0.27299178
 -0.04389238  0.09126669 -0.01078604 -0.19103487  0.16208621  0.24604802
 -0.07307502  0.          0.         -0.12718023  0.24403808  0.28584546
 -0.18264626  0.00165479 -0.06684495  0.00174129  0.04332867 -0.23543346
 -0.21512655 -0.03737333 -0.21452675 -0.00089282  0.02823531  0.03758622
 -0.1417162  -0.06118829  0.14893324  0.15240117 -0.41867969  0.02062737
 -0.20171128 -0.05199453 -0.02878504 -0.05690205 -0.00822025  0.16697299
  0.05883053 -0.02389096  0.00136731 -0.16762332 -0.028629   -0.32861881
  0.03634985 -0.07389198  0.21275861  0.04526623 -0.02054795  0.06962269
  0.10974472 -0.00829821  0.02919795  0.02919795  0.13425862 -0.1541579
 -0.14939502  0.07194375 -0.05521903  0.00808614  0.25223741]
c1: 0.36137368435839057
c2: 0.1470695705640188
c3: -0.289264094943774
v: [ 5.42277772  4.3876175  -1.70228648]
d: -4.842660198868449
```

1.3 Part C: Concepts

1.3.1 Question 1

- In B2, what values of weights in the final layer were chosen? What does this tell you about the three regressions you fit to make the "cheater" w's?
  - The weights for the final layer were 6.17, 4.19, -1.1, and d = -4.34. The first hidden node has the most weight, essentially meaning it has the most influence on the output, where the second and third have moderate to little influence.
  - This tells us that the first regression was the most important, indicating that the effects features were the most predictive of the cannabis type. And then the flavor features were much more predictive than the rating features. This is also pretty consistent with our understanding of the data, as the effects are the most indicative of the type of cannabis, followed by the flavor, and then the rating.

1.3.2 Question 2

- Use your fitted model from B4 to predict on the data, and make a confusion matrix. How did this model perform compare to our "cheater" one?

[154]:
```python
from sklearn.metrics import confusion_matrix

# Predict using the fitted parameters
def predict_with_fitted_model(preds, w1, w2, w3, c1, c2, c3, v, d):
    preds = np.array(preds)
    z1 = np.dot(preds, w1) + c1
    z2 = np.dot(preds, w2) + c2
    z3 = np.dot(preds, w3) + c3
    u1 = sigmoid(z1)
    u2 = sigmoid(z2)
    u3 = sigmoid(z3)
    u = np.array([u1, u2, u3])
    # hidden layer
    hidden_layer_input = np.dot(u, v) + d
    output = sigmoid(hidden_layer_input)
    return output * 100

# Predict on the data
predictions = [predict_with_fitted_model(p, w1, w2, w3, c1, c2, c3, v, d) for p␣
 ↪in cannabis_preds]

predictions = [1 if p > 50 else 0 for p in predictions]

# Compute confusion matrix
conf_matrix = confusion_matrix(true_labels, predictions)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[619  68]
 [ 50 381]]
```

[156]:
```
# cheater model confusion matrix
train_predicted = cannabis[effect_cols.tolist() + flavors_cols.tolist() +␣
 ↪['Rating']]
preds = []
for i in range(len(train_predicted)):
    effects_preds = train_predicted.iloc[i][effect_cols]
    flavors_preds = train_predicted.iloc[i][flavors_cols]
    rating_pred = train_predicted.iloc[i]['Rating']

    # supress warnings
    import warnings
    warnings.filterwarnings("ignore")
    p_effects = m1.predict_proba([effects_preds])[0][1]
    p_flavors = m2.predict_proba([flavors_preds])[0][1]
    p_rating = m3.predict_proba([[rating_pred]])[0][1]

    p = (p_effects + p_flavors + p_rating) / 3
    preds.append(1 if p > 0.5 else 0)

cheater_conf_matrix = confusion_matrix(true_labels, preds)
print("Cheater Confusion Matrix:")
print(cheater_conf_matrix)
```

```
Cheater Confusion Matrix:
[[656  31]
 [147 284]]
```

The fitted model did perform better compared to my "cheater" one. The fitted model has less true positives but it has more true negatives, which means the accuracy would be higher for the fitted model. From the confusion matrix, it seems that the model was better when the actual type was indica and worse when predicting when the actual type was sativa.