CSE4088 – Introduction to Machine Learning – HW2

Huzeyfe Ayaz

150119700

Generalization Error

Q1.)

$$P[|E_{in}(g) - E_{out}(g)| > 0.05] \leq 2e^{-2(0.05)^2N} \leq 0.03$$

$$\Rightarrow e^{-2(0.05)^2N} \leq 0.015 \Rightarrow -2(0.05)^2N \leq \ln(0.015)$$

$$\Rightarrow N \geq \frac{\ln(0.015)}{-0.005} \Rightarrow N \geq 839.94 \quad \boxed{B \rightarrow 1000}$$

Q2.)

→ Same as the previous question but we should divide the inside of ln by 10

$$N \geq \frac{\ln(0.0015)}{-0.005} \Rightarrow N \geq 1300.45 \quad \boxed{C \rightarrow 1500}$$

Q3.)

$$N \geq \frac{\ln(0.00015)}{-0.005} \Rightarrow N \geq 1760.975 \quad \boxed{D \rightarrow 2000}$$

The Perceptron Learning Algorithm

Q4.)

```
iterations = [] # to store the # of iterations in each step
model = PLA()    # initializing a PLA model
errors = []
for i in range(1000):
    X, f, y = create_data_and_pick_target_function()
    X_test, _, y_test = create_data_and_pick_target_function(sample_siz
e=1000, target_function=f)

    iteration = model.fit(X, y)
    y_pred = model.predict(X_test)

    iterations.append(iteration)
    errors.append(1 - accuracy_score(y_test, y_pred))

np.mean(iterations)
```

Output: `10.378`

In question 4, I have created a list to store the number of iterations in each step and initialized a PLA model. X, f, and y represents the data, target function and true labels respectively in the form of Numpy array. 'create_data_and_pick_target_function' function has default values as sample_size=10, feature_size=2, and target_function=None so that we can create 10 sample with 2 features and a target function randomly.

Q5.)

The solution of question 5 is included in question 4. I have created 1000 test (Eout) sample to evaluate my model. Then, I have got the predictions using predict method. In order to calculate the errors, I simple take the 1 – accuracy_score that implemented in scikit-learn library (it takes the counts of matched labels and divides to the number of total labels to find accuracy score). As an output, I got the following results:

```
np.mean(errors)
```

Output: `0.10647200000000001`

Q6.)

```
iterations = []
model = PLA()
errors = []
for i in range(1000):
    X, f, y = create_data_and_pick_target_function(sample_size=100)
    X_test, _, y_test = create_data_and_pick_target_function(sample_siz
e=1000, target_function=f)

    iteration = model.fit(X, y)
    y_pred = model.predict(X_test)

    iterations.append(iteration)
    errors.append(1 - accuracy_score(y_test, y_pred))

np.mean(iterations)
```

Output:
```
101.283
```

Q7.)

I applied the same steps in question 5 except the trained model is different. For this case, I trained a new model with 100 sample and then created 1000 test sample to evaluate model. The final results are:

```
np.mean(errors)
```

Output:
```
0.013936000000000011
```

Linear Regression

Q8.)

```python
errors = []
model = LinearRegression()
for i in range(1000):
    X, f, y = create_data_and_pick_target_function(sample_size=100)
    model.fit(X, y)
    y_pred = model.predict(X)
    errors.append(1 - accuracy_score(y, y_pred))

np.mean(errors)
```

Output:
```
0.03847000000000001
```

In this case, I have created an instance of LinearRegression model to perform classification task. After fitting the model, I got the prediction of training dataset (Ein) to find the in-sample error. Lastly, average error of the 1000 iteration is computed.

Q9.)

```python
errors = []
for i in range(1000):
    X, f, y = create_data_and_pick_target_function(sample_size=1000, target_function=f)
    y_pred = model.predict(X)
    errors.append(1 - accuracy_score(y, y_pred)) # error = 1 - accuracy

np.mean(errors)
```

Output:
```
0.007672000000000007
```

For question 9, Test data (Eout) with 1000 sample is randomly generated to find the out-sample error. To perform that operation, the fitted model that generated in previous question is used to predict the labels of Test data.

Q10.)

```python
reg_model_w = model.w
iterations = []
model = PLA(weight_type='custom', weights=reg_model_w)
for i in range(1000):
    X, f, y = create_data_and_pick_target_function(sample_size=10, target_function=f)
    iteration, error = model.fit(X, y)
    iterations.append(iteration)

np.mean(iterations)
```

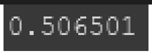
Output:
```
1.439
```

In this case, I take the weights of previous model to provide a custom weight to PLA model. Then, PLA is fitted over those weights instead of creating a new zero weights.

Non-Linear Transformation:

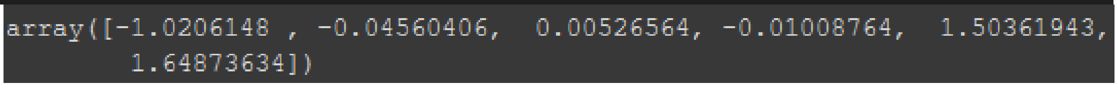
Q11.)

```
def flip_y(y):
    rand_choices = np.random.choice(np.arange(y.shape[0]), 100)
    y[rand_choices] *= -1
    return y
```
Flip function randomly chooses 100 sample which is 10% of 1000 sample and flips their signs.

```
errors = []
model = LinearRegression()
for i in range(1000):
    X, f, y = create_data_and_pick_target_function(1000, target_functio
n="sign_determinant")
    y = flip_y(y) ←
    model.fit(X, y)
    y_pred = model.predict(X)
    errors.append(1 - accuracy_score(y, y_pred))

np.mean(errors)
```
Output: 
```
0.506501
```

As opposed to previous questions, I have changed the sign of some labels as indicated by green arrow and fit the model with some noisy dataset. Errors are found like in previous questions.

Q12.)

```
model = LinearRegression()

X, f, y = create_data_and_pick_target_function(sample_size=1000, target
_function="sign_determinant")
X = np.concatenate([X,
                    (X[:, 1]*X[:, 2]).reshape(-1,1),
                    (X[:, 1]**2).reshape(-1,1),
                    (X[:, 2]**2).reshape(-
1,1)], axis=1) # concatenating all features
y = flip_y(y)
model.fit(X, y)

model.w
```
Output: 
```
array([-1.0206148 , -0.04560406,  0.00526564, -0.01008764,  1.50361943,
        1.64873634])
```

To perform question 12, I added 3 more features to our dataset as X1*X2, X1**2, and X2**2. After all, I concatenate them with my original dataset using the concatenate function of numpy library. Then flipped the labels of 100 sample and fit the model with new data. The result of weights is printed as above.

Q13.)

```
errors = []
for i in range(1000):
    X, f, y = create_data_and_pick_target_function(sample_size=1000, target_function="sign_determinant")
    X = np.concatenate([X,
                        (X[:, 1]*X[:, 2]).reshape(-1,1),
                        (X[:, 1]**2).reshape(-1,1),
                        (X[:, 2]**2).reshape(-1,1)], axis=1)
    y = flip_y(y)
    y_pred = model.predict(X)
    errors.append(1 - accuracy_score(y, y_pred))

np.mean(errors)
```

Output:
```
0.12599000000000002
```

For question 13, the steps are same as question 12. However, in here I calculated the in-sample errors as mentioned in question.