0ras3csxj

November 9, 2024

CS589 ASSIGNMENT 6

Name: Dorian Benhamou Goldfajn
Email: dbenhamougol@umass.edu
Discussed With: Aryan Nair

[2]:
```python
import numpy as np

stuff=np.load("data.npz")
X_trn = stuff["X_trn"]
y_trn = stuff["y_trn"]
X_tst = stuff["X_tst"]
# no Y_tst !
```

Question 1

[100]:
```python
X = [1.0, 2.0, 3.0, 4.0, 5.0]
Y = [1, 1, 0, 0, 1]
splits = [0.5, 1.5, 2.5, 3.5, 4.5, 5.5]

# calculate parent entropy

parent_p1 = sum(Y) / len(Y)
parent_p0 = 1 - parent_p1
parent_entropy = (-parent_p1*np.log2(parent_p1)) + (- parent_p0*np.
 ↪log2(parent_p0))

# calculate child entropy for each split
child_entropy = {.5: -1, 1.5: -1, 2.5: -1, 3.5: -1, 4.5: -1, 5.5: -1}

for split in splits:
    # split data
    left = []
    right = []
    for i in range(len(X)):
        if X[i] < split:
            left.append(Y[i])
        else:
```

```python
            right.append(Y[i])

    # calculate I(p1) and I(p2) for both left and right sides (if empty, set to 0)
    left_p1 = 0
    if len(left) != 0:
        left_p1 = sum(left) / len(left)
    left_p0 = 1 - left_p1

    right_p1 = 0
    if len(right) != 0:
        right_p1 = sum(right) / len(right)
    right_p0 = 1 - right_p1

    if left_p1 == 0:
        left_entropy = -left_p0*np.log2(left_p0)
    elif left_p0 == 0:
        left_entropy = -left_p1*np.log2(left_p1)
    else:
        left_entropy = (-left_p1*np.log2(left_p1)) + (- left_p0*np.log2(left_p0))

    if right_p1 == 0:
        right_entropy = -right_p0*np.log2(right_p0)
    elif right_p0 == 0:
        right_entropy = -right_p1*np.log2(right_p1)
    else:
        right_entropy = (-right_p1*np.log2(right_p1)) + (- right_p0*np.log2(right_p0))

    # calculate child entropy and store
    child_entropy[split] = ((len(left))*left_entropy + (len(right))*right_entropy) / len(Y)

# calculate information gain and print
information_gain = {split: parent_entropy - child_entropy[split] for split in splits}

print("Split | Information Gain")
for split in splits:
    print(f" {split}  |  {information_gain[split]}")
```

```
Split | Information Gain
 0.5  |  0.0
 1.5  |  0.17095059445466854
 2.5  |  0.4199730940219748
 3.5  |  0.01997309402197489
```

```
4.5  |  0.17095059445466854
5.5  |  0.0
```

Question 2

```
[101]: class ClassificationStump():
           def __init__(self):
               return

           def fit(self, X_trn, y_trn):

               # do stuff here
               D = len(X_trn[0])
               N = len(X_trn)
               dim = -1
               thresh = -1
               min_error = np.inf
               c_left = 0
               c_right = 0

               for i in range(D):

                   sorted_indices = np.argsort(X_trn[:, i])
                   Z = X_trn[sorted_indices]
                   y_sorted = y_trn[sorted_indices]


                   for n in range(N-1):
                       t = (Z[n][i] + Z[n+1][i])/2

                       R1 = y_sorted[:n+1] # x <= t
                       R2 = y_sorted[n+1:] # x > t

                       R1_count = np.bincount(R1, minlength=np.max(y_sorted) + 1)
                       R2_count = np.bincount(R2, minlength=np.max(y_sorted) + 1)

                       c1 = np.argmax(R1_count)
                       c2 = np.argmax(R2_count)

                       p1 = R1_count / len(R1) # probability of each class in R1
                       p2 = R2_count / len(R2) # probability of each class in R2

                       gini_left = np.sum(p1 * (1-p1)) # sum pi(1-pi) for each class␣
        ↪in R1

                       gini_right = np.sum(p2 * (1-p2))

                       gini_total = (len(R1) / N) * gini_left + (len(R2) / N) *␣
        ↪gini_right # weighted average of gini impurity
```

```
                error = gini_total
                if error < min_error:
                    min_error = error
                    dim = i
                    c_left = c1
                    c_right = c2
                    thresh = t

        self.model = (dim, thresh, c_left, c_right)
        self.model = {'dim': dim, 'thresh': thresh, 'c_left': c_left, 'c_right':
 ↪ c_right}
        return

    def predict(self, X_val, y_val=None):
        assert hasattr(self, "model"), "No fitted model!"
        # do stuff here (use self.model for prediction)
        y_pred = []
        for x_val in X_val:
            y_pred.append(self.model['c_left'] if x_val[self.model['dim']] <=␣
 ↪self.model['thresh'] else self.model['c_right'])

        return y_pred
```

[102]:

```
data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']
X_tst = data['X_tst']

clf = ClassificationStump()


X_trn = X_trn.reshape((6000, 3 * 29 * 29))


clf.fit(X_trn, y_trn)
y_pred = clf.predict(X_trn)
correct_pred = sum(y_pred == y_trn)
e = 1 - correct_pred / len(y_pred)
print("Training classification error: ", e)
```

```
Training classification error:  0.6421666666666667
```

Question 3

[103]:

```
# train classifcation trees
from sklearn.tree import DecisionTreeClassifier
```

```
data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']

X_trn = X_trn.reshape((len(X_trn), 3 * 29 * 29))

max_depths = [1,3,6,9,12,14]
# return a 6x1 table of classification errors
print(" DEPTH | TRAINING ERROR")
for max_depth in max_depths:
    clf = DecisionTreeClassifier(max_depth=max_depth)
    clf.fit(X_trn, y_trn)
    y_pred = clf.predict(X_trn)
    correct_pred = sum(y_pred == y_trn)
    acc = correct_pred / len(y_trn)
    e = 1 - acc
    print(f"  {max_depth}  | {e}")
```

```
 DEPTH | TRAINING ERROR
  1   | 0.6433333333333333
  3   | 0.5515
  6   | 0.4033333333333333
  9   | 0.2493333333333333
  12  | 0.1293333333333333
  14  | 0.07150000000000001
```

Question 4

Order of O(M). For each depth in the tree, we check a single threshold value (O(1)) until we get to the end of the tree where we make final classication, giving rise O(M * 1) = O(M).

Question 5

O(D * Nlogn). We must iterate through each D, and for each D we sort N items and iterate through N items. This is equivalent to O(D * (NlogN + N)) = O(D * NlogN), although I think it slightly depends on implementation.

Question 6

[104]:
```
# linear classifier model with logistic loss and ridge regularization only␣
 ↪using sklearn.linear and decision_function() method

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']
X_tst = data['X_tst']
```

```
X_trn = X_trn.reshape((6000, 3*29*29))


lambda_vals = [0.1, 1, 10, 100, 1000]

# report training classification error and logistic loss for each lambda value
print(" LAMBDA | TRAINING ERROR | LOGISTIC LOSS")
for lambda_val in lambda_vals:
    clf = LogisticRegression(penalty='l2', C=1/lambda_val, max_iter=10000, tol=.
 ↪001)
    clf.fit(X_trn, y_trn)

    confidence_scores = clf.decision_function(X_trn) # get confidence score per␣
 ↪class per sample

    exp_scores = np.exp(confidence_scores) # prepare for softmax
    sum_exp_scores_per_sample = [sum(scores) for scores in exp_scores]

    probs = [exp_scores[i] / sum_exp_scores_per_sample[i] for i in␣
 ↪range(len(exp_scores))] # softmax to get probs
    y_pred = [np.argmax(prob) for prob in probs] # get prediction from highest␣
 ↪prob
    logistic_loss = log_loss(y_trn, probs) # calculate logistic loss

    correct_preds = np.sum(y_pred == y_trn)
    e = 1 - (correct_preds / len(y_trn)) # calculate classification error
    print(f"  {lambda_val}  | {e} | {logistic_loss}")
```

```
LAMBDA | TRAINING ERROR | LOGISTIC LOSS
 0.1  | 0.17000000000000004 | 0.4771340095214954
 1  | 0.21383333333333332 | 0.582162089440951
 10  | 0.26449999999999996 | 0.6939481457195188
 100  | 0.3031666666666667 | 0.7863802988369172
 1000  | 0.33666666666666667 | 0.877910964337898
```

Question 7

[105]:
```
# K nearest neighbors classifier
from sklearn.neighbors import KNeighborsClassifier

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']
X_tst = data['X_tst']

X_trn = X_trn.reshape((len(X_trn), 3 * 29 * 29))
```

```
k_neighbors = [1, 3, 5, 7, 9, 11]

print(" K | TRAINING ERROR")
for k in k_neighbors:
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_trn, y_trn)
    y_pred = clf.predict(X_trn)
    correct_pred = sum(y_pred == y_trn)
    acc = correct_pred / len(y_trn)
    e = 1 - acc
    print(f" {k} | {e}")
```

```
 K | TRAINING ERROR
 1 | 0.0
 3 | 0.29200000000000004
 5 | 0.31633333333333336
 7 | 0.3478333333333333
 9 | 0.3626666666666667
 11 | 0.3663333333333333
```

Question 8

Order of O(N * (N * D + NlogN)). For each of x1,..., xN values in X_tst, we get the distance for each D for each x in X_trn (N * (N * D)). Per x in X_tst, we must also sort the distances to get K nearest neighbor and select majority class, giving rise to (N * ((N * D) + NlogN + K + K)), but since K is usally much smaller than N, we can abstract it to the order mentioned above.

PART 2 - Cross Validation

Question 9

[4]:
```
from sklearn.model_selection import KFold

class Classifier():
    def __init__(self, model):
        self.model = model
        return

    def fit(self, X_trn, y_trn):
        self.model.fit(X_trn, y_trn)
        # self.model is stored
        return

    def predict(self, X_val, y_val=None):
        # self.model is used
        y_pred = self.model.predict(X_val)
        return y_pred
```

```python
def cross_validation(classifier, X_trn, y_trn, n_folds=5):
    # do stuff here

    kf = KFold(n_splits=n_folds)
    outputs = []

    for train_index, test_index in kf.split(X_trn):

        x_train, x_test = X_trn[train_index], X_trn[test_index]
        y_train, y_test = y_trn[train_index], y_trn[test_index]


        classifier.fit(x_train, y_train)
        y_pred = classifier.predict(x_test)
        correct_preds = np.sum(y_pred == y_test)
        e = 1 - (correct_preds / len(y_test))

        outputs.append((classifier, e))

    # Return the paired (model and error) for all folds
    return outputs # [(model1, error1), (model2, error2), ..., (modelK, errorK)]
```

Question 10

[5]:
```python
# Usage for the cross_validation function:
from sklearn.tree import DecisionTreeClassifier
import pandas as pd

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']

X_trn = X_trn.reshape((6000, 3 * 29 * 29))

results = []

# Perform cross-validation for different max_depth values
for max_depth in [1, 3, 6, 9, 12, 14]:
    classifier = Classifier(DecisionTreeClassifier(max_depth=max_depth))
    N = 5
    outputs = cross_validation(classifier, X_trn, y_trn, n_folds=N)

    # Collect the errors for each fold
    fold_errors = [out_[1] for out_ in outputs]
    avg_error = np.mean(fold_errors)

    # Append the results to the list
```

```
    results.append([max_depth] + fold_errors + [avg_error])

# Create a DataFrame from the results
columns = ["MAX_DEPTH", "FOLD_1_ERROR", "FOLD_2_ERROR", "FOLD_3_ERROR",␣
↪"FOLD_4_ERROR", "FOLD_5_ERROR", "AVG_ERROR"]
df = pd.DataFrame(results, columns=columns)

# Display the DataFrame as a table
print(df.to_string(index=False))
```

```
 MAX_DEPTH  FOLD_1_ERROR  FOLD_2_ERROR  FOLD_3_ERROR  FOLD_4_ERROR  FOLD_5_ERROR
AVG_ERROR
         1      0.654167      0.661667      0.635833      0.637500      0.630000
0.643833
         3      0.527500      0.569167      0.565000      0.547500      0.540000
0.549833
         6      0.486667      0.493333      0.497500      0.482500      0.480833
0.488167
         9      0.460000      0.499167      0.470833      0.464167      0.463333
0.471500
        12      0.480833      0.489167      0.464167      0.463333      0.479167
0.475333
        14      0.470000      0.510000      0.478333      0.457500      0.472500
0.477667
```

Question 11

[6]:
```
from sklearn.linear_model import LogisticRegression
import pandas as pd

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']

X_trn = X_trn.reshape((6000, 3 * 29 * 29))


# Initialize a list to store the results
results = []

# Perform cross-validation for different lambda values
for lambda_val in [0.1, 1, 10, 100, 1000]:
    classifier = Classifier(LogisticRegression(penalty='l2', C=1/lambda_val,␣
↪max_iter=10000, tol=.001))
    N = 5
    outputs = cross_validation(classifier, X_trn, y_trn, n_folds=N)

    # Collect the errors for each fold
```

```python
    fold_errors = [out_[1] for out_ in outputs]
    avg_error = np.mean(fold_errors)

    # Append the results to the list
    results.append([lambda_val] + fold_errors + [avg_error])

# Create a DataFrame from the results
columns = ["LAMBDA", "FOLD_1_ERROR", "FOLD_2_ERROR", "FOLD_3_ERROR",␣
 ↪"FOLD_4_ERROR", "FOLD_5_ERROR", "AVG_ERROR"]
df = pd.DataFrame(results, columns=columns)

# Display the DataFrame as a table
print(df.to_string(index=False))
```

```
 LAMBDA  FOLD_1_ERROR  FOLD_2_ERROR  FOLD_3_ERROR  FOLD_4_ERROR  FOLD_5_ERROR
AVG_ERROR
    0.1      0.458333      0.484167      0.482500      0.472500      0.480833
0.475667
    1.0      0.422500      0.427500      0.435000      0.428333      0.412500
0.425167
   10.0      0.377500      0.407500      0.392500      0.397500      0.353333
0.385667
  100.0      0.342500      0.379167      0.348333      0.361667      0.327500
0.351833
 1000.0      0.350833      0.374167      0.352500      0.357500      0.340000
0.355000
```

Question 12

[8]:
```python
class KNNClassifier():
    def __init__(self, n_neighbors):
        self.n_neighbors = n_neighbors
        return

    def fit(self, X_trn, y_trn):
        # Just store X_trn, y_trn
        self.model = (X_trn, y_trn)

    def predict(self, X_val, y_val=None):
        assert hasattr(self, "model"), "No fitted model!"
        # do stuff here (use self.model for prediction)

        def KNN_predict_(X_trn, y_trn, x, K):
            # Dictionary to store n: distance pairs
            distances = {}
            for n in range(len(X_trn)):
                # Calculate distance between x and x[n]
                distance = np.sqrt(np.sum((x - X_trn[n])**2))
```

```
                distances[n] = distance

            # Sort distances in ascending order
            sorted_distances = sorted(distances.items(), key=lambda x: x[1])

            # Get the K nearest neighbors
            y_neighbor = []
            for n in range(K):
                n_nearest = sorted_distances[n][0]
                y_neighbor.append(y_trn[n_nearest])

            # get majority class
            c_pred = np.argmax(np.bincount(y_neighbor))
            return c_pred

        X_trn, y_trn = self.model
        y_pred = []
        for x in X_val:
            y_pred.append(KNN_predict_(X_trn, y_trn, x, self.n_neighbors))

        return y_pred
```

[9]:
```
k_neighbors = [1, 3, 5, 7, 9, 11]

# Initialize a list to store the results
results = []

# Perform cross-validation for different k values
for k in k_neighbors:
    knn = KNNClassifier(k)
    knn.fit(X_trn, y_trn)
    models = cross_validation(knn, X_trn, y_trn, n_folds=5)

    # Collect the errors for each fold
    fold_errors = [model[1] for model in models]
    avg_error = np.mean(fold_errors)

    # Append the results to the list
    results.append([k] + fold_errors + [avg_error])

# Create a DataFrame from the results
columns = ["K", "FOLD_1_ERROR", "FOLD_2_ERROR", "FOLD_3_ERROR", "FOLD_4_ERROR",␣
 ↪"FOLD_5_ERROR", "AVG_ERROR"]
df = pd.DataFrame(results, columns=columns)

# Display the DataFrame as a table
print(df.to_string(index=False))
```

```
 K  FOLD_1_ERROR  FOLD_2_ERROR  FOLD_3_ERROR  FOLD_4_ERROR  FOLD_5_ERROR
AVG_ERROR
 1       0.470000       0.458333       0.457500       0.440833       0.457500
0.456833
 3       0.482500       0.458333       0.465833       0.454167       0.486667
0.469500
 5       0.455000       0.441667       0.451667       0.432500       0.465000
0.449167
 7       0.438333       0.429167       0.446667       0.426667       0.448333
0.437833
 9       0.433333       0.425833       0.433333       0.433333       0.445000
0.434167
11       0.430833       0.435000       0.440833       0.427500       0.430833
0.433000
```

Question 13

Model | hyper-parameters | cross-validation avg. error | public leaderboard accuracy \
Classification Tree | max_depth = 9 | .474 | .57866 \
Classification Tree | max_depth = 7 | .4743 | x\
\
Logistic Regression | lambda = 10 | .388 | .62133\
Logistic Regression | lambda = 100 | .352 | x\
\
KNN Classification | K = 7 | 0.438 | .55733\
KNN Classification | K = 9 | 0.434 | x\
KNN Classification | K = 11 | 0.434 | x\

THE FIRST MODEL WAS USED FOR INITIAL SUBMISSION, EXPERIMENTING WITH THE REST AS WELL AS I ACTUALLY THINK BEST PERFORMANCES WOULD BE: Tree with max depth of 9, logistic regression with lambda = 100, KNN with K = 9 or 11.

[ ]:
```
import numpy as np
import csv

def write_csv(y_pred, filename):
    """Write a 1d numpy array to a Kaggle-compatible .csv file"""
    with open(filename, 'w') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(['Id', 'Category'])
        for idx, y in enumerate(y_pred):
            csv_writer.writerow([idx, y])
```

[117]:
```
# Train best tree model on test data and save predictions

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']
```

```python
X_tst = data['X_tst']

X_trn = X_trn.reshape((len(X_trn), 3 * 29 * 29))
X_tst = X_tst.reshape((len(X_tst), 3 * 29 * 29))

clf = DecisionTreeClassifier(max_depth = 9)
clf.fit(X_trn, y_trn)
y_pred = clf.predict(X_tst)
write_csv(y_pred, 'tree_predictions_9.csv')
```

[118]:
```python
# Train best logistic regression model on test data and save predictions

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']
X_tst = data['X_tst']

X_trn = X_trn.reshape((len(X_trn), 3 * 29 * 29))
X_tst = X_tst.reshape((len(X_tst), 3 * 29 * 29))

lambda_val = 100
clf = LogisticRegression(penalty='l2', C=1/lambda_val, max_iter=10000, tol=.001)
clf.fit(X_trn, y_trn)
y_pred = clf.predict(X_tst)
write_csv(y_pred, 'logistic_predictions_100.csv')
```

[121]:
```python
# Train best k nearest neighbor on test data and save predictions

data = np.load('data.npz')
X_trn = data['X_trn']
y_trn = data['y_trn']
X_tst = data['X_tst']

X_trn = X_trn.reshape((len(X_trn), 3 * 29 * 29))
X_tst = X_tst.reshape((len(X_tst), 3 * 29 * 29))

k = 9
clf = KNNClassifier(k)
clf.fit(X_trn, y_trn)
y_pred = clf.predict(X_tst)
write_csv(y_pred, 'knn_predictions_9.csv')
```

Question 14

Private Scoring Predictions:

1) Tree Classification model: I believe the error range would be similar to the error present in the public set and slightly lower than cross validations errors, as depth of 9 should be balanced

to avoid overfitting while trying to minimize error on unseen data. The public set seems to be performing better than cross validation predicted, indicating slightly better performance overall which I'd expect to continue in private set.

2) Logistic Regression model: The cross validation classification error is very similar to that of public test set, indicating a decent prediction job by cross validation. I'd expect a similar error range in the private test set, with room for slight variations.

3) KNN Classification model: I believe the error would slightly increase on the private set as the performance in public set seems to perform worse than cross validation predicted, indicating that the model seems to struggle more than expected and I would expect the trend to continue in private set.