Lilyanne Kurth-Yontz

Assignment 2 - Report

I chose to center this assignment on k nearest neighbors, perceptron, random forests, support vector machines, and neural network algorithms. All five are classification-based algorithms.

The original training file is 40,000 rows by 101 columns. Splitting this into vectors and labels gives us a 40,000x100 vector array and a 40,000x1 label array. Similarly, the original testing file is 10,000 rows by 101 columns. When split into vectors and labels, we get a 10,000x100 vector array and a 10,000x1 label array. The vectors represented here are based on the word2vec model, which is a shallow neural network that saves each unique word it encounters as a vector. Since the training and testing files are based on movie review data, we can assume that each vector in the array represents a unique word that someone typed up in their movie review.

My k nearest neighbors model used 5 as the number of neighbors (represented in scikit-learn as the “n_neighbors” parameter), and Euclidean distance as the distance measure (represented as the “metric” parameter). Time elapsed increases as k increases, and time elapsed varies between distance measures. In this case, the model took roughly 101 seconds to run. The resulting accuracy was around 75% -- not fantastic, but okay.

On the other hand, my perceptron model used 100 as n, the maximum number of passes over the training data (represented in scikit-learn as the “max_iter” parameter). Like k in the k nearest neighbors algorithm, time elapsed increases as n increases, but this isn’t a problem

because perceptrons are already extremely fast. For example, this model took roughly 3 seconds to run, and it returned a higher accuracy than the k nearest neighbors model -- around 82%.

For my own experiments, I chose random forests, SVM, and neural networks. I chose these three algorithms mainly because of my prior experience with them in COGS 118A, which I had taken this spring. I found these three algorithms to be the most interesting -- plus, they are often ranked among the best. I decided to see if they lived up to their standards.

For random forests, I chose to modify the parameter "n_estimators" (number of trees in the forest) from 10 to 50. My initial choice was between 10, 50, and 100 trees. 10 trees gave me an accuracy score of around 74%, so that was not satisfactory. 100 trees returned roughly the same accuracy score as 50 trees, but took twice as long. Therefore, for the sake of efficiency, I chose 50 trees. This model gave me an accuracy score of around 81%, and it took around 16 seconds to run. (For comparison, the 100 trees model gave an accuracy score of around 82%, and it took around 32 seconds to run.)

For my support vector machines, I chose to modify the "kernel" parameter from "rbf" to "poly", and I modified the "degree" parameter (specifically for polynomial kernels) from 3 to 1. My primary issue with this model was that it took a very, very long time to run. Linear kernels are generally fastest, but are less accurate. On the other hand, radial basis function (RBF) kernels are the most accurate, but are the slowest. Polynomial kernels lie somewhere in between. I started with a linear kernel, but the accuracy score was almost 10% below the desired accuracy score of 80% . So, I went with a polynomial kernel, and reduced the degree from 3 to 1, in the hope of reducing the time elapsed. Ultimately, this model gave me the highest accuracy score of the three I'd chosen (around 87%), but it gave me the longest time elapsed, too (around 101 seconds).

For my neural network (or as scikit-learn calls it, “multi-layer perceptron classifier”), I chose to modify the “hidden_layer_sizes” parameter from 100 to 5 and the “max_iter” parameter from 200 to 100. (“hidden_layer_sizes” refers to the number of neurons in a given layer, and “max_iter” refers to the maximum number of iterations.) When I initially used this classifier with the default parameters, it took even longer than my support vector machines model. I figured some of the largest default values could be trimmed to reduce the time elapsed, so I diminished the values of “hidden_layer_sizes” and “max_iter”. At first, I reduced “max_iter” to 50, but I received a runtime message saying that convergence was not guaranteed with this small of a value. So, I increased it to 100, and after that, no message appeared. This model was by far the most efficient -- it achieved the same accuracy as my support vector machines model (87%) while taking only about 4 seconds to run.

Ultimately, each of these three models has its pros and cons. Random forests were a decently fast algorithm to work with, but their accuracy was the lowest of the three. Plus, it would be easy for new users to misunderstand “n_estimators” and think that accuracy increases as the parameter increases. This is definitely not the case -- there is a “sweet spot” that one has to reach, which means the user has to take time “pruning” their trees to ensure the highest accuracy with the lowest time cost. Support vector machines provided one of the highest accuracies, but had a tremendous time cost. This dataset wasn’t even that large to begin with, but it still took almost two minutes to run. It is highly accurate, but it is highly frustrating, too. Neural networks gave the best accuracy coupled with the best time, but it took a lot of tuning to get there. The scikit-learn page for the MLPClassifier() has a long list of parameters that aren’t always intuitive. Recall how the neural networks took a very long amount of time at first, but it was only after I tuned the parameters that I received such efficient results. So, this is definitely not a “beginner-friendly” algorithm.

Overall, this assignment has a lot to reveal about classifier algorithms. “Simple” algorithms such as perceptrons may be perfect for the task, while more “advanced” algorithms like random forests and SVMs are not always as good as expected. Different machine learning algorithms are best suited for different tasks, and for this one, perceptrons and neural networks served the best.

SentimentAnalysis

September 7, 2019

0.1 Assignment 2

Feel free to add cells to define your helper functions.

Please remove "raise NotImplementedError()" Student 1 Name: Lilyanne Kurth-Yontz
Student 2 Name: -

Lilyanne Kurth-Yontz and - attest that this assignment was done by them two and reflects their original work and based on their understanding of the concepts. Both students have equally contributed to the solution of this assignment.

```
In [16]: import numpy as np
         import sklearn
         import time
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.linear_model import Perceptron
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.svm import SVC
         from sklearn.neural_network import MLPClassifier
         import os
```

```
In [17]: def vector_label_split(read_file):
             '''

             This method takes care of splitting the input file into vector arrays and label a
             Called in readTrainTestData().

             Input: read_file (file to be read)

             Returns: vectors (array of vectors, rxc)
                      labels (array of labels, rx1)

             '''

             vectors = []
             labels = []
```

```
    for line in read_file:

        value = '' #number or label to be added
        label_acquired = False

        vec_to_add = []
        lab_to_add = []

        line_length = len(line)
        point_in_line = 0

        for char in line:
            if char == ' ':
                if not label_acquired:
                    lab_to_add.append(value) #append to labels
                    label_acquired = True
                else:
                    vec_to_add.append(value) #append to vectors
                value = ''
            else:
                if point_in_line == line_length - 1:
                    vec_to_add.append(value) #append last value to vectors
                else:
                    value += char
                point_in_line += 1

        labels.append(lab_to_add)
        vectors.append(vec_to_add)

    return vectors, labels
```

In [18]:
```
def readTrainTestData(trainFile, testFile):
    '''
    Input: trainFile - name of training data file (str)
           testFile - name of test data file (str)

    Returns: train_vec - training data variables (list of arrays)
             train_lab - training data labels (list)
             test_vec - test data variables (list of arrays)
             test_lab - test data labels (list)
    '''

    # YOUR CODE HERE

    train = open(trainFile, 'r')
    test = open(testFile, 'r')

    train_vec, train_lab = vector_label_split(train)
```

```
    test_vec, test_lab = vector_label_split(test)

    train.close()
    test.close()

    #convert lists to numpy arrays to access .shape
    train_vec = np.asarray(train_vec)
    train_lab = np.asarray(train_lab)
    test_vec = np.asarray(test_vec)
    test_lab = np.asarray(test_lab)

    return train_vec, train_lab, test_vec, test_lab
```

In [19]:
```
train_vec, train_lab, test_vec, test_lab = readTrainTestData('train_data.txt', 'test_

assert(len(train_vec) == 40000)
assert(train_vec[0].shape == (100,))
assert(len(train_lab) == 40000)

assert(len(test_vec) == 10000)
assert(test_vec[0].shape == (100,))
assert(len(test_lab) == 10000)
```

In [20]:
```
def sentimentClassKNN(train_vec, train_lab, test_vec, test_lab, k, distanceMeasure):
    # Create a KNN model
    # Fit the training set into the model
    # Calculate and return the prediction accuracy on test set

    # YOUR CODE HERE

    #start timer
    start = time.time()

    #convert to float for easier processing
    train_lab = train_lab.astype(float)
    test_lab = test_lab.astype(float)
    train_vec = train_vec.astype(float)
    test_vec = test_vec.astype(float)

    #knn
    knn = KNeighborsClassifier(n_neighbors = k, metric = distanceMeasure)
    knn.fit(train_vec, train_lab)
    test_predict = knn.predict(test_vec)
    accuracy = accuracy_score(test_predict, test_lab)

    print("Accuracy: " + str(accuracy))

    #end timer
```

```
    end = time.time()
    timer = end - start

    print("Seconds passed:", timer)

    return accuracy
```

```
In [21]: knn_accu = sentimentClassKNN(train_vec, train_lab, test_vec, test_lab, 5, "euclidean")
         assert (0.70<knn_accu)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:19: DataConversionWarning: A colu
```

```
Accuracy: 0.7455
Seconds passed: 103.91123509407043
```

```
In [22]: def sentimentClassPerceptron(train_vec, train_lab, test_vec, test_lab, n):
    # Create a Perceptron model
    # Fit the training set into the model
    # Calculate and return the prediction accuracy on test set

    # YOUR CODE HERE

    #start timer
    start = time.time()

    #convert to float, since perceptron only accepts numerical values
    train_lab = train_lab.astype(float)
    test_lab = test_lab.astype(float)
    train_vec = train_vec.astype(float)
    test_vec = test_vec.astype(float)

    #perceptron
    perc = Perceptron(max_iter = n)
    perc.fit(train_vec, train_lab)
    test_predict = perc.predict(test_vec)
    accuracy = accuracy_score(test_predict, test_lab)

    print("Accuracy: " + str(accuracy))

    #end timer
    end = time.time()
    timer = end - start

    print("Seconds passed:", timer)

    return accuracy
```

```
In [23]: per_accu = sentimentClassPerceptron(train_vec, train_lab, test_vec, test_lab, 100)
         assert (0.75 < per_accu)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataConversionWarning:
  y = column_or_1d(y, warn=True)
```

```
Accuracy: 0.8218
Seconds passed: 3.229156017303467
```

```
In [24]: def sentimentClassMyOwn(train_vec, train_lab, test_vec, test_lab):
             """
             Try three different models
             """
             # Create three different models
             # Fit the training set into three models
             # Return prediction accuracy on test set

             # YOUR CODE HERE

             #convert to float for easier processing
             train_lab = train_lab.astype(float)
             test_lab = test_lab.astype(float)
             train_vec = train_vec.astype(float)
             test_vec = test_vec.astype(float)

             '''

             RANDOM FOREST CLASSIFIER
             n_estimators (number of trees) = 50

             '''

             #start timer
             start = time.time()

             #random forest
             forest = RandomForestClassifier(n_estimators = 50)
             forest.fit(train_vec, train_lab)
             test_predict = forest.predict(test_vec)
             accuracy1 = accuracy_score(test_predict, test_lab)

             print("Accuracy 1: " + str(accuracy1))

             #end timer
             end = time.time()
             timer1 = end - start
```

```
print("Seconds passed:", timer1)

'''

SVM
kernel = 'poly' (polynomial)
degree (of polynomial kernel) = 1

'''

#start timer
start = time.time()

#SVM
svm = SVC(kernel = 'poly', degree = 1)
svm.fit(train_vec, train_lab)
test_predict = svm.predict(test_vec)
accuracy2 = accuracy_score(test_predict, test_lab)

print("Accuracy 2: " + str(accuracy2))

#end timer
end = time.time()
timer2 = end - start

print("Seconds passed:", timer2)

'''

NEURAL NETWORK
hidden_layer_sizes (number of neurons in ith layer) = 5
max_iter (maximum number of iterations until convergence) = 100

'''

#start timer
start = time.time()

#neural network
nn = MLPClassifier(hidden_layer_sizes = 5, max_iter = 100)
nn.fit(train_vec, train_lab)
test_predict = nn.predict(test_vec)
accuracy3 = accuracy_score(test_predict, test_lab)

print("Accuracy 3: " + str(accuracy3))

#end timer
```

```
            end = time.time()
            timer3 = end - start

            print("Seconds passed:", timer3)

            return accuracy1, accuracy2, accuracy3
```

In [25]:
```
accu1, accu2, accu3 = sentimentClassMyOwn(train_vec, train_lab, test_vec, test_lab)
assert (0.8 < accu1)
assert (0.8 < accu2)
assert (0.8 < accu3)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:29: DataConversionWarning: A colu


Accuracy 1: 0.8091
Seconds passed: 15.847410202026367


/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataConversionWarning:
  y = column_or_1d(y, warn=True)


Accuracy 2: 0.8692
Seconds passed: 102.03818964958191


/opt/conda/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:912: Da
  y = column_or_1d(y, warn=True)


Accuracy 3: 0.8654
Seconds passed: 4.39799165725708
```

In [26]:
```
### Check to make sure you include your report.
assert os.path.isfile('Assignment2_Report.pdf')
```