**Project 3**

**Members**: Weston Rainer, Reece Wienandt, Ganesh Budhathoki, Noah Fields

Weston Rainer worked on the genetic algorithm and the report. Reece Wienandt worked on the decision tree code and the corresponding report. Noah Fields worked with Weston on the genetic algorithm and the report.

During the decision tree part of the project it uses the sklearn decision tree classifier, so the program retrieves the data off of the training csv file, prepares it for input into the decision tree classifier, such as filling missing entries, or converting to integers. After training data has been added it creates the table, then input the scraped and prepared data for the test, and it prints out the results to a csv file. It also renders the table generated in a pdf file in the same directory as the source code. There was two major difficulties with the assignment. First was understanding how the sklearn import really worked, which I found out required the data without the column I was looking for, then you give it that column as a “target”. The second was preparing the data for the algorithm in a way that both works, but doesn’t manipulate or cause the data to be unreliable. On a scale of 1-10 the difficulty for decision tree approach using imports is a 2.

1. Briefly describe the steps involved in applying the ID3 decision tree for this classification task.

You need to compute and compare the information gain for each property, of which I didn’t use certain properties such as name, and cabin since names don’t determine survival rate, and cabin information was only available for specific individuals. After getting the highest information gain, it would be listed as the first branch in the decision tree and each branch would calculate the newest information gain for the next best property.

2. What is the total number of nodes in the tree?

406

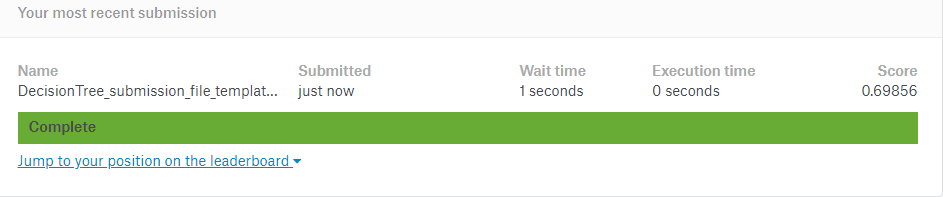
3. What is the total number of leaf nodes in the tree?

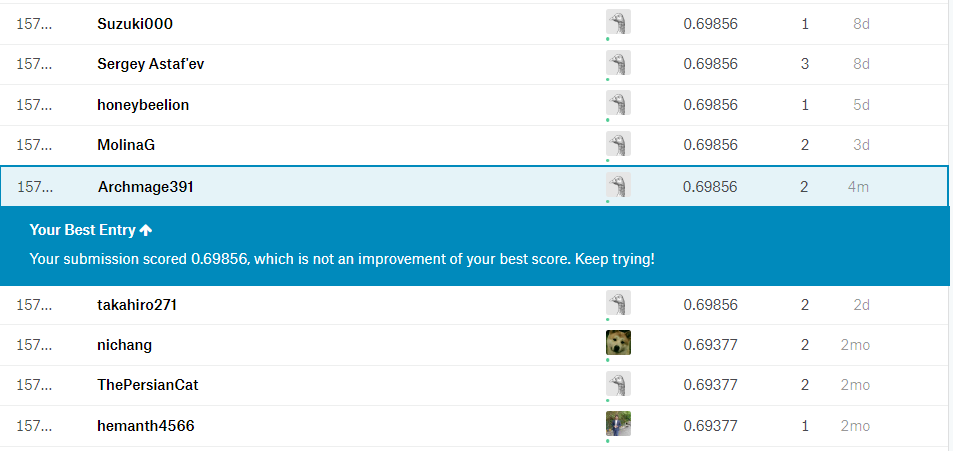
204





4. What are the classification accuracy on the training set and the test set, respectively?





For the genetic algorithm it begins with a string that is the code that we want to put into our genetic algorithm and an array with an expected output from the code. We create an initial population of three which is then mutated and ranked. At which point the next generation is created of six. Afterwards each offspring is mutated and ranked. Once ranked the three highest are chosen to make the next generation. When the code is being mutated there is a one in three chance that it will be mutated. If it decides it will be mutated then it chooses to either mutate the line order, the variables or the operations of assignment. If line order is chosen to be mutated then it decides picks two random lines to exchange. If the variables are to be replaced it finds all of the variables in the code and replaces the variable on some lines. If the assignment operation is chosen it randomly decides which type of operation to exchange on a random line. To score a function the expected output needs to be passed in. At which point the mutated code gets a point for each correct output and if the output is in the correct order. In the crossover function the three best ones are selected and half of the lines of code are combined to create the next generation. The function will terminate when it reaches a maximum score or if it reaches an upper limit of iterations.

We ran into a few problems while working on the genetic algorithm. A couple of the problems were how to determine if it was an assignment operation and how to mutate only a specific line of code. We fixed this by splitting the string into lines and then to find out if it was an assignment operation all we had to do is see if the second value in the line contained an =. To find the variables we struggled to determine what the word of the variable was and what type they were. We solved this by looking at the initial assignment and seeing if it contained a ”” as then we could tell if it was a string. On a scale from 1-10 we would rate the difficulty level at a 5.

Ganesh Budhathoki worked on the Support Vector Machine algorithm and applying SVM to MNIST dataset which uses kernel trick to transform data to find optimal boundary between outputs. For the start of this algorithm, given MNIST dataset was used with 10,000 samples. The first process of this algorithm was data preprocessing where actual data pixel was separated from data label/classifier. Then, the dataset was separated into test and train dataset model which is run through Support Vector Classifier to transform the input data into linear, polynomial, sigmoid form to find the optimal boundary between the dataset. And finally the trained data was used to predict the accuracy of test data model.

In the process prediction were made to calculate the test accuracy dataset based on varying range of train dataset sample (between 20-80), and varying kernel trick (linear, polynomial or sigmoid). Comparing the three kernel trick (linear, polynomial and sigmoid), polynomial transformation of dataset produced high test accuracy on all range of dataset sample i.e 96% accuracy , being the highest on 80% train dataset and sigmoid produced the lowest test accuracy at 86% on 80% train dataset

Some of the sample runs are given below:

Avg test accuracy for polynomial transformed data with 50% train dataset:

accuracy 0.94 5000

macro avg 0.95 0.94 0.94 5000

weighted avg 0.94 0.94 0.94 5000

Avg test accuracy for linear transformed data with 50% train dataset

accuracy 0.92 5000

macro avg 0.92 0.92 0.92 5000

weighted avg 0.92 0.92 0.92 5000

SVM classifies data and transform them using kernel trick to find better boundary okay

It is hard to compare the three algorithms as they all accomplish different tasks. The Decision Tree is good at finding a way to classify large sets of data while the SVM is good at classifying data and transforming it using different kernel trick like into linear, polynomial or sigmoid form and depending on data set it will find the optimal boundary between output solution. This helps to improve the result of output solutions as it helps to find the optimal classification of data for train and test data model. While the genetic algorithm is meant to try to improve something over time with random mutation.

For project three we implemented three algorithms, the ID-3 decision tree, the Support Vector Machine, and a Genetic Algorithm that would try to create a working program with a correct output. After completing these three tasks we now have a better idea of how to create a program that utilizes AI algorithms and how they could possibly be implemented in the real world. Furthermore, we have a deeper understanding of the different methods in which an AI can function/operate, we feel that this was a good conclusion to the semester.

**SVM**

**Part 2. Evaluate and answer the following questions [15 points]**

Then answer the following questions:

i. Briefly describe the steps involved in applying the SVM for this digits classification task.

Answer:

Data preprocessing: Firstly, data is processed and separated into classification data and real pixel data. and then the real data is further split into train and test data in the program.

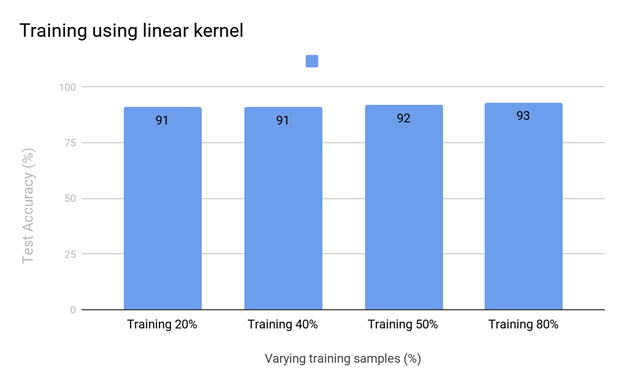
Train Algorithm: After the data is preprocessed, we further separate them into train data set and test data set. The we use different kernels are used to transform data into required form like linear, polynomial, etc. And then finally we train the model using two train data sets.

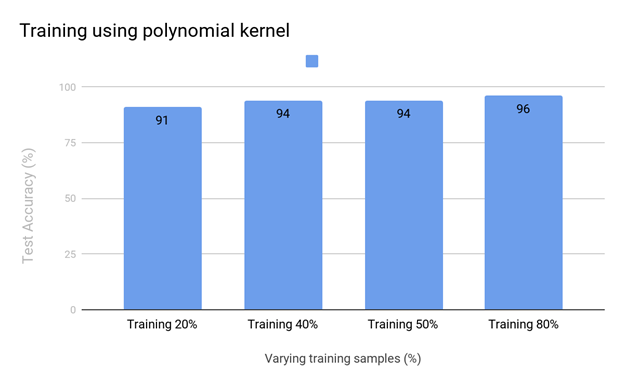
Prediction and Evaluation: After training the model we feed the test model to report on accuracy of predication of the model and its classification. If test report is poor, we can manually change the input data into different kernel type for better data transformation or better classification to get optimal solution.

Finally, the accuracy is compared.

ii. Train your SVM with increasing sizes of training set, say 10%, 20%, …. Test it with the test set. Make a plot to show how training and test accuracies vary with number of training samples.

Answer:





iii. Test your SVM using different types of SVM kernels, training them using increasing sizes of training set, say 10%, 20%, …. Test the trees with the test set. Make plots to show how training and test accuracies vary with number of training samples.

Answer:

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.svm *import* SVC

*from* sklearn.metrics *import* classification\_report, confusion\_matrix

#read in csv file

svmData = pd.read\_csv('../mnist\_subset.csv')

shape = svmData.shape

head = svmData.head()

#Separation of data labels and acutal data pixel

dataPixel = svmData.iloc[:,1:]

classifier = svmData.iloc[:, 0]

# dataPixel = svmData.drop('7', axis=1)

# classifier = svmData['7']

#separating training and test dataset

#using various sample % for testing and training data ranging from 20%, 30%..

#using stratify to preserve uniform distribution of labels

train\_1, test\_1, train\_2, test\_2 = train\_test\_split(dataPixel, classifier, train\_size=0.2, stratify=classifier)

#using linear kernel

# svmDataClassifier = SVC(kernel='linear')

#using poly kernel

svmDataClassifier = SVC(kernel='poly')

#training model for best fit

svmDataClassifier.fit(train\_1, train\_2)

#making prediction

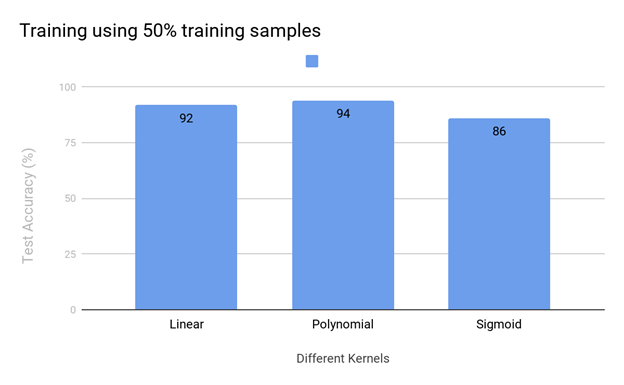
predictionTest2 = svmDataClassifier.predict(test\_1)

# print confusion matix

print(confusion\_matrix(test\_2, predictionTest2))

#print accuracy report

print(classification\_report(test\_2, predictionTest2))



GENETIC ALGORITHM

import random  
import re  
from operator import itemgetter  
  
  
def categorize\_variables*(*codeArray*)*:  
 variables = *{*"string": *[]*,  
 "number": *[]*,  
 "variable": *[]*,  
 "array": *[]  
 }* for line in codeArray:  
 if '=' in line:  
 line = re.sub*(*'\s+', '', line*)* variable\_name, value = line.split*(*'='*)* string\_rules = *[*'\'' in value,  
 '"' in value,  
 '\'' in value,  
 value not in variables*[*"string"*]]* number\_rules = *[*bool*(*re.search*(*r'\d', value*))* == True,  
 not all*(*string\_rules*)*,  
 value not in variables*[*"number"*]  
 ]* array\_rules = *[*value*[*0*]* == '[',  
 value not in variables*[*"array"*]  
 ]* variable\_rules = *[*not all*(*string\_rules*)*,  
 bool*(*re.search*(*'*[*a-zA-Z*]*', value*))* == True,  
 value not in variables*[*"variable"*]  
 ]* if all*(*number\_rules*)*:  
 variables*[*"number"*]*.append*(*variable\_name*)* elif all*(*string\_rules*)*:  
 variables*[*"string"*]*.append*(*variable\_name*)* elif all*(*variable\_rules*)*:  
 variables*[*"variable"*]*.append*(*variable\_name*)* elif all*(*array\_rules*)*:  
 variables*[*"array"*]*.append*(*variable\_name*)* return variables  
  
  
def swap\_variables*(*codeArray*)*:  
 variables = categorize\_variables*(*codeArray*)* output = *[]* for line in codeArray:  
 decider = random.randint*(*0, 1*)* if decider >= 1:  
 for key, value in variables.items*()*:  
 if any*(*var in value for var in line*)*:  
 matching = *[*s for s in value if s in line*]* for var in matching:  
 line = line.replace*(*var, random.choice*(*value*))* output.append*(*line*)* return output  
  
  
def mutate*(*code*)*:  
 codeArray = code.splitlines*()* if random.randint*(*1, 3*)* == 1:  
 mutation = random.randint*(*1, 3*)* if mutation == 1:  
 codeArray = change\_order*(*codeArray*)* if mutation == 2:  
 codeArray = swap\_variables*(*codeArray*)* if mutation == 3:  
 codeArray = ChangeAssignmentOperation*(*codeArray*)* return '\n'.join*(*str*(*e*)* for e in codeArray*)*def Crossover*(*parents*)*:  
 genes = *{*"top": *[]*, "bottom": *[]}* children = *[]* for parent in parents:  
 end = parent.count*(*'\n'*)* + 1  
 middle = int*(*end / 2*)* genes*[*'top'*]*.append*(*parent.splitlines*()[*0:middle*])* genes*[*'bottom'*]*.append*(*parent.splitlines*()[*middle:end*])* for r in range*(*0, len*(*genes*[*"top"*]))*:  
 for j in range*(*0, len*(*genes*[*"bottom"*]))*:  
 if r != j:  
 children.append*(*'\n'.join*(*genes*[*"top"*][*r*]* + genes*[*"bottom"*][*j*]))* return children  
  
  
def ChangeAssignmentOperation*(*codeArray*)*:  
 for i in range*(*0, len*(*codeArray*))*:  
 assignmentCheck = codeArray*[*i*]*.split*(*" "*)* if len*(*assignmentCheck*)* > 1:  
 if assignmentCheck*[*1*]* == "=" or assignmentCheck*[*1*]* == "+=" or assignmentCheck*[*1*]* == "-=" or assignmentCheck*[* 1*]* == "/=" or assignmentCheck*[*1*]* == "\*=":  
 decider = random.randint*(*0, 11*)* if decider == 0:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"+", "\*"*)* if decider == 1:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"+", "/"*)* if decider == 2:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"+", "-"*)* if decider == 3:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"-", "\*"*)* if decider == 4:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"-", "/"*)* if decider == 5:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"-", "+"*)* if decider == 6:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"/", "\*"*)* if decider == 7:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"/", "+"*)* if decider == 8:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"/", "-"*)* if decider == 9:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"\*", "+"*)* if decider == 10:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"\*", "/"*)* if decider == 11:  
 codeArray*[*i*]* = codeArray*[*i*]*.replace*(*"\*", "-"*)* return codeArray  
  
  
def change\_order*(*codeArray*)*:  
 swapOne = random.randint*(*0, len*(*codeArray*)* - 1*)* swapTwo = random.randint*(*0, len*(*codeArray*)* - 1*)* temp = codeArray*[*swapOne*]* codeArray*[*swapOne*]* = codeArray*[*swapTwo*]* codeArray*[*swapTwo*]* = temp  
 return codeArray  
  
  
# noinspection PyUnresolvedReferences  
def fitness*(*code, anwser*)*:  
 score = 0  
 arrayTest = *[]* try:  
 #NEED TO CHANGE THIS WHEN TESTING OTHER FUNCTIONS  
 arrayTest = makeArray*()* for i in range*(*0, len*(*arrayTest*))*:  
 if arrayTest*[*i*]* == answer*[*i*]*:  
 score += 1  
 if answer*[*1*]* in arrayTest:  
 score += 1  
 except:  
 print*(*"Unexpected error:"*)* score = -1  
 return score  
  
def results*(*iteration, gen\_high, max\_score, highest\_score, highest\_score\_gen, gen\_size, pass\_count, fail\_count, codes = *[])*:  
 if*(*len*(*codes*)* > 0*)*:  
 print*(*"\n\nFinal Generation Code: "*)* for x in codes:  
 print*(*x, end="\n\n"*)* print*(*"\n\nFinal Results: "*)* print*(*"\nIteration: {0} \  
 \n\tHighest Score in Current Gen: {1} / {2} \  
 \n\tHighest Total Score: {3} (Gen {4}) \  
 \n\tGen Info (Size, Pass, Fail): {5}, {6}, {7}"  
 .format*(*iteration, gen\_high, max\_score, highest\_score, highest\_score\_gen, gen\_size, pass\_count, fail\_count*))*code = """def makeArray():  
\tarray = []  
\tx = 0 + 1  
\tarray.append(x)  
\tx = 2 - 2  
\tarray.append(x)  
\tx = 4 \* 2  
\tarray.append(x)  
\treturn array"""  
  
  
  
answer = *[*0, 4, 2*]*codes = *[]*random.seed*()*codes.append*(*code*)*codes.append*(*code*)*codes.append*(*code*)*offspring\_per\_pop = 6  
max\_score = len*(*answer*)* \* 2  
max\_iterations = 1000  
highest\_score = *[*0, 1*]*iteration = 0  
while True:  
 iteration += 1  
 gen\_high = 0  
 ranks = *{}* fail\_count = 0  
 gen\_size = len*(*codes*)* for i in range*(*0, gen\_size*)*:  
 score = 0  
 codes*[*i*]* = mutate*(*codes*[*i*])* try:  
 exec*(*codes*[*i*])* score = fitness*(*codes*[*i*]*, answer*)* except Exception as e:  
 fail\_count += 1  
 score = -1  
 ranks*[*i*]* = score  
 if score >= gen\_high:  
 gen\_high = score  
 top\_3 = sorted*(*ranks.keys*()*, key=ranks.get, reverse=True*)[*:3*]* pass\_count = gen\_size - fail\_count  
  
 if gen\_high > highest\_score*[*0*]*:  
 highest\_score*[*0*]*, highest\_score*[*1*]* = gen\_high, iteration  
 if highest\_score*[*0*]* >= max\_score or max\_iterations <= iteration:  
 results*(*iteration, gen\_high, max\_score, highest\_score*[*0*]*, highest\_score*[*1*]*, gen\_size, pass\_count, fail\_count, codes*)* break  
 else:  
 results*(*iteration, gen\_high, max\_score, highest\_score*[*0*]*, highest\_score*[*1*]*, gen\_size, pass\_count, fail\_count*)* codes = Crossover*(*list*(*itemgetter*(*\*top\_3*)(*codes*)))*#########################################################  
################TEST CASES BELOW#########################  
#########################################################  
  
###INIT SEQUENTIAL###  
# def Sequential\_Search(dlist, item):  
# \tfound = False  
# \twhile pos < len(dlist) and not found:  
# \t\tif dlist[pos] == item:  
# \t\t\tfound = True  
# \t\telse:  
# \t\t\tpos = pos + 1  
# \tpos = 0  
# \treturn found, pos  
#  
# Sequential\_Search([11, 23, 58, 31, 56, 77, 43, 12, 65, 19], 31))  
# Correct Output: (True, 3)  
#  
# FINAL RESULT  
# Iteration: 5  
# Highest Score in Current Gen: 20 / 20  
# Highest Total Score: 20 (Gen 5)  
# Gen Info (Size, Pass, Fail): 6, 6, 0  
#  
# BEST SEQUENTIAL  
# def Sequential\_Search(dlist, item):  
# \tpos = 0  
# \tfound = False  
# \twhile pos < len(dlist) and not found:  
# \t\tif dlist[pos] == item:  
# \t\t\tfound = True  
# \t\telse:  
# \t\t\tpos = pos + 1  
# \treturn found, pos  
  
  
  
  
###INIT BUBBLE###  
# def bubbleSort():  
# \tarr = [64, 34, 25, 12, 22, 11, 90]  
# \tfor i in range(n):  
# \t\tfor j in range(0, n - i - 1):  
# \t\t\tif arr[j] > arr[j + 1]:  
# \t\t\t\tarr[j], arr[j + 1] = arr[j + 1], arr[j]  
# \tn = len(arr)  
# \treturn arr  
#  
# bubbleSort([64, 34, 25, 12, 22, 11, 90])  
# Correct Output: [11, 12, 22, 25, 34, 64, 90]  
#  
# FINAL RESULT  
#  
# Iteration: 1000  
# Highest Score in Current Gen: 8 / 14  
# Highest Total Score: 8 (Gen 414)  
# Gen Info (Size, Pass, Fail): 6, 6, 0  
  
# BEST BUBBLE  
# def bubbleSort():  
# \tarr = [64, 34, 25, 12, 22, 11, 90]  
# \tn = len(arr)  
# \tfor i in range(n):  
# \t\tfor j in range(0, n - i - 1):  
# \t\t\tif arr[j] > arr[j + 1]:  
# \t\t\t\tarr[j], arr[j + 1] = arr[j + 1], arr[j]  
# \treturn arr  
  
  
  
  
###INIT MAKEARRAY###  
# def makeArray():  
# \tarray = []  
# \tx = 0 + 1  
# \tarray.append(x)  
# \tx = 2 - 2  
# \tarray.append(x)  
# \tx = 4 \* 2  
# \tarray.append(x)  
# \treturn array  
#  
# Correct: [0, 4, 2]  
#  
# FINAL RESULT  
#  
# Iteration: 30  
# Highest Score in Current Gen: 6 / 6  
# Highest Total Score: 6 (Gen 30)  
# Gen Info (Size, Pass, Fail): 6, 6, 0  
#  
# BEST MAKEARRAY  
# def makeArray():  
# x = 0 \* 1  
# array = []  
# array.append(x)  
# x = 2 \* 2  
# array.append(x)  
# x = 4 / 2  
# array.append(x)  
# return array