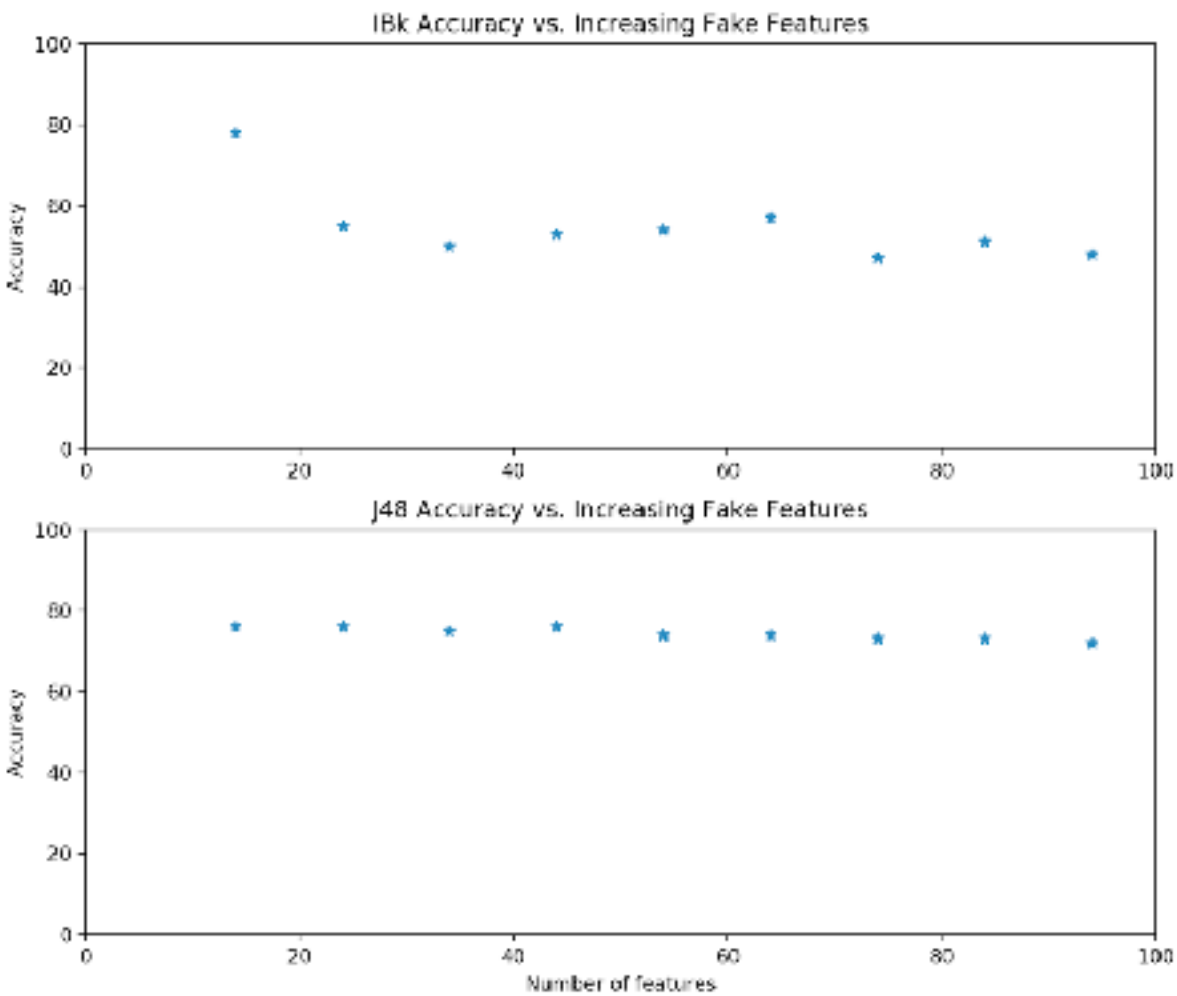
Morgan Ciliv

Empirical/Programming Assignment 1: Discussion

The sensitivity, or True Positive rate, probability of detection or recall is a measure of the proportion of positives that are identified correctly.

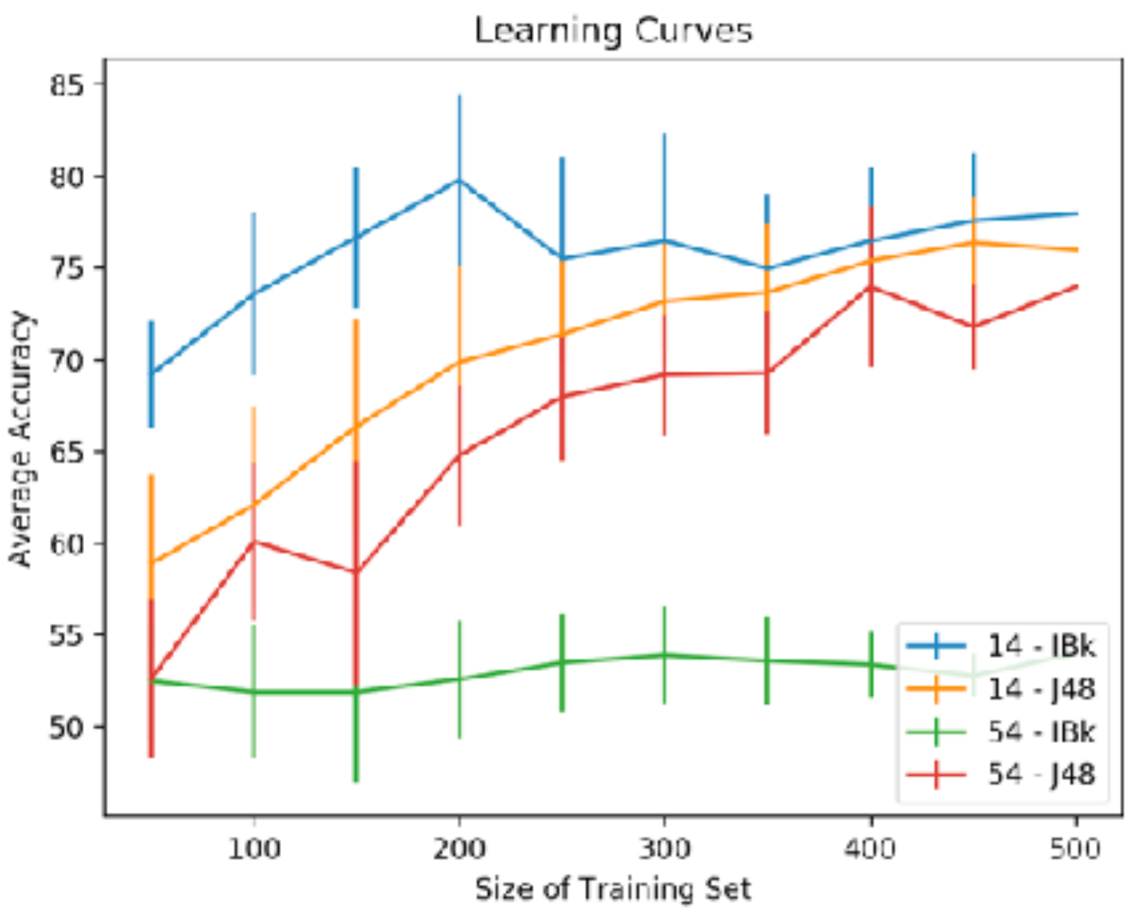
Part 1:

From part 1, based on the correlations in the graph below, we can hypothesize that the increase of fake features affects algorithm J48 less while IBk is more susceptible to this kind of hinderance in the data. We can conclude that J48 is more sensitive with regard to features specifically.



Part 2:

By looking at the trends in the Learning Curves graph below, we can see that the algorithms tend to improve with more training examples, which makes sense because more samples lead to higher confidence in the trend. However, IBk with fake features didn’t increase accuracy with extra training examples, which indicates that it was not as sensitive in detecting the relationships between the right features. Again, this shows us that J48 is a more sensitive algorithm.



References:

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

The code, kNN_decTree.py:

```
# -----------------------------------------------------------------------------
# By: Morgan Ciliv
# Discussed work with Vincent Tsang and Zach Kirsch
#
# COMP 135 - Machine Learning - Fall 2017
# Empircal/Programming Assignment 1
# -----------------------------------------------------------------------------

import subprocess
import re
import matplotlib.pyplot as plt
import random
import shutil
import numpy
import math

ALGOS = ["IBk", "J48"]
NUM_SAMPLES = 10
TRAIN_SET_SIZES = range(50, 501, 50)

# -----------------------------------------------------------------------------
# Functions (For part 1 and 2)
# -----------------------------------------------------------------------------

def get_weka_output(algo, train_file, test_file):
    if algo == "IBk":
        algo_str = "lazy.IBk"
    elif algo == "J48":
        algo_str = "trees.J48"
    else:
        print "Algorithm not realized."
        exit(2)

    command = "{}{}{}{}{}{}".format("java weka.classifiers.", algo_str,
                                    " -t ", train_file, " -T ", test_file)
    return subprocess.check_output(command, shell=True)

def get_weka_accuracy(str_output):
    return float(re.findall(r'Correctly\sClassified\sInstances\s+\d+\s+(\d+)',
                            str_output)[1])

# -----------------------------------------------------------------------------
# Part 1
# -----------------------------------------------------------------------------

num_features = range(14,95,10)
accuracies1 = {ALGOS[0]: [], ALGOS[1]: []}
for num in num_features:
    for algo in ALGOS:
        train_file = "$WEKADATA/EEGTrainingData_" + str(num) + ".arff"
```

```
        test_file = "$WEKADATA/EEGTestingData_" + str(num) + ".arff"
        output = get_weka_output(algo, train_file, test_file)
        accuracies1[algo].append(get_weka_accuracy(output))

# Plotting
plt.figure(1)

plt.subplot(211)
plt.title("IBk Accuracy vs. Increasing Fake Features")
plt.ylabel("Accuracy")
plt.axis([0, 100, 0, 100])
plt.plot(num_features, accuracies1["IBk"], '*')

plt.subplot(212)
plt.title("J48 Accuracy vs. Increasing Fake Features")
plt.xlabel("Number of features")
plt.ylabel("Accuracy")
plt.axis([0, 100, 0, 100])
plt.plot(num_features, accuracies1["J48"], '*')

plt.show()

# --------------------------------------------------------------------------------
# Part 2
# --------------------------------------------------------------------------------

# Removes endline characters
def read_file(file_name):
    with open(file_name) as file:
        file_lines = file.readlines()
    file_lines = [line.strip() for line in file_lines]
    return file_lines

# Writes training file back out with new name
def write_file(num_train_examples, new_data_for_file):
    filename = "train" + str(num_train_examples) + ".arff"
    with open(filename, 'w+') as file:
        for line in new_data_for_file:
            line = line + "\n"
            file.write(line)
    return filename

# Do for 14 and 54 features
trials = [14, 54]
sample_accuracies_by_trial = {trials[0]: [], trials[1]: []}
avgs_by_trial = {trials[0]: [], trials[1]: []}
stddevs_by_trial = {trials[0]: [], trials[1]: []}

for trial in trials: # TODO: Change
    sample_accuracies = []

    # Identify header and example portions
    train_orig = "{}{}{}".format("data/EEGTrainingData_", trial, ".arff")
    train_file_lines = read_file(train_orig)
```

```
    for i, line in enumerate(train_file_lines):
        if line == "@DATA":
            data_line = i + 1
    pre_data = train_file_lines[:data_line]
    train_data = train_file_lines[data_line:]

    testing_orig = "{}{}{}".format("data/EEGTestingData_", trial, ".arff")
    shutil.copyfile(testing_orig, "test.arff")

    # Repeat 10 times for 10 samples
    for i in range(NUM_SAMPLES):
        # Randomly shuff the order of the examples
        random.shuffle(train_data)

        accuracies2 = {ALGOS[0]: [], ALGOS[1]: []}
        for num_train_examples in TRAIN_SET_SIZES:
            # Put the initial segment of i examples (in  permuted order) from
            # train.arff into traini.arff
            subset_train_data = train_data[:num_train_examples]
            new_data_for_file = pre_data + subset_train_data
            train_file = write_file(num_train_examples, new_data_for_file)

            # Run the learning algorithm on the corresponding train/test
            # combination and record the accuracy on the test data

            for algo in ALGOS:
                output = get_weka_output(algo, train_file, "test.arff")
                accuracies2[algo].append(get_weka_accuracy(output))

        sample_accuracies.append(accuracies2)
    sample_accuracies_by_trial[trial] = sample_accuracies

    # Calculate the avg and std dev in performance over the 10 trials
    avgs = {ALGOS[0]: [], ALGOS[1]: []}
    stddevs = {ALGOS[0]: [], ALGOS[1]: []}
    for algo in ALGOS:
        for i, _ in enumerate(TRAIN_SET_SIZES):
            sum = 0.0
            for j in range(NUM_SAMPLES):
                sum += sample_accuracies[j][algo][i]
            avg = sum / float(NUM_SAMPLES)
            avgs[algo].append(avg)

            sqdiffs = 0.0
            for j in range(NUM_SAMPLES):
                sqdiffs += (sample_accuracies[j][algo][i] - avg) ** 2
            stddev = math.sqrt(sqdiffs / (NUM_SAMPLES - 1)) # Of sample
            stddevs[algo].append(stddev)
    avgs_by_trial[trial] = avgs
    stddevs_by_trial[trial] = stddevs

# Plotting
plt.figure(2)
plt.title("Learning Curves")
```

```
plt.ylabel("Average Accuracy")
plt.xlabel("Size of Training Set")
a = plt.errorbar(TRAIN_SET_SIZES, avgs_by_trial[14]['IBk'],
          yerr=stddevs_by_trial[14]['IBk'])
b = plt.errorbar(TRAIN_SET_SIZES, avgs_by_trial[14]['J48'],
          yerr=stddevs_by_trial[14]['J48'])
c = plt.errorbar(TRAIN_SET_SIZES, avgs_by_trial[54]['IBk'],
          yerr=stddevs_by_trial[54]['IBk'])
d = plt.errorbar(TRAIN_SET_SIZES, avgs_by_trial[54]['J48'],
          yerr=stddevs_by_trial[54]['J48'])
plt.legend([a, b, c, d], ["14 - IBk", "14 - J48", "54 - IBk", "54 - J48"],
         loc=4)
plt.show()
```