Alex Weeden

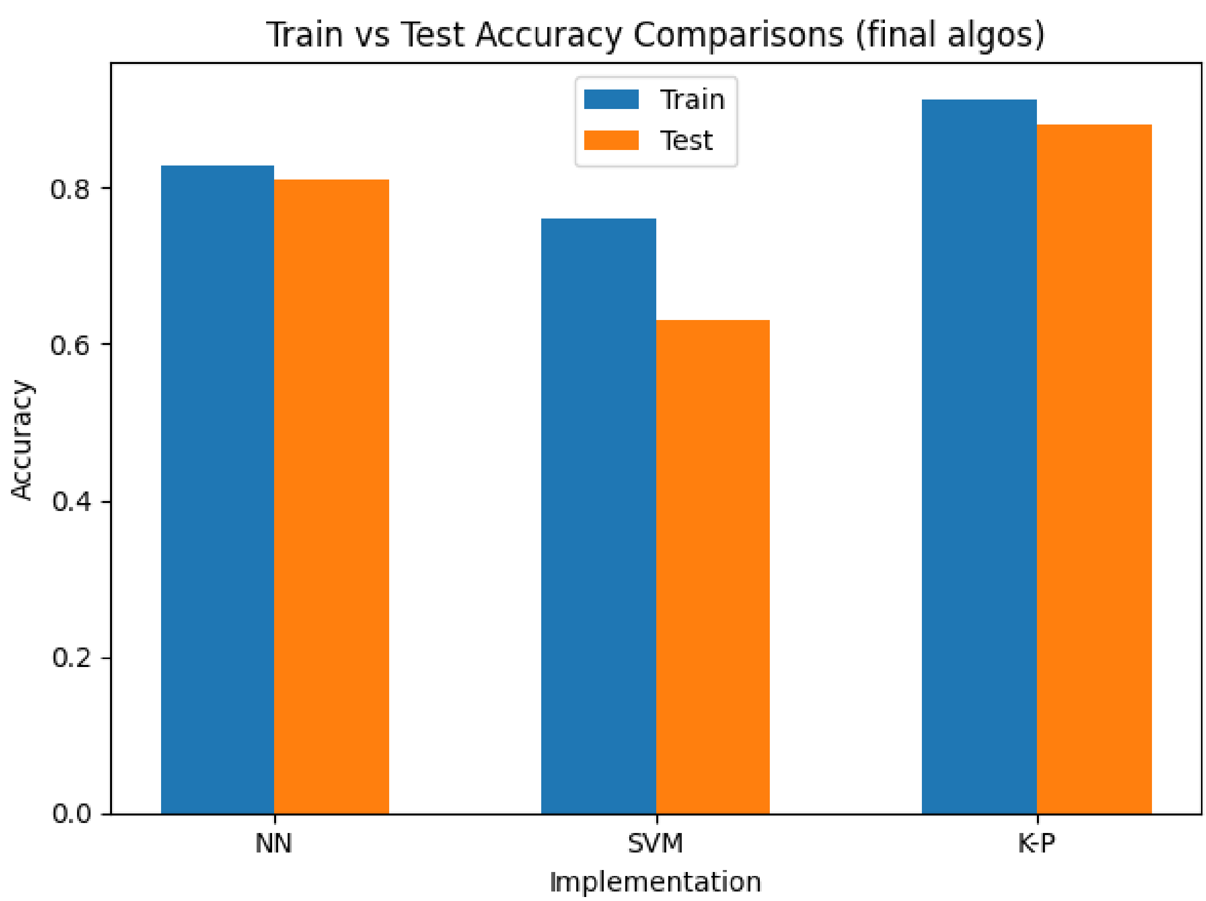
Ajw0057

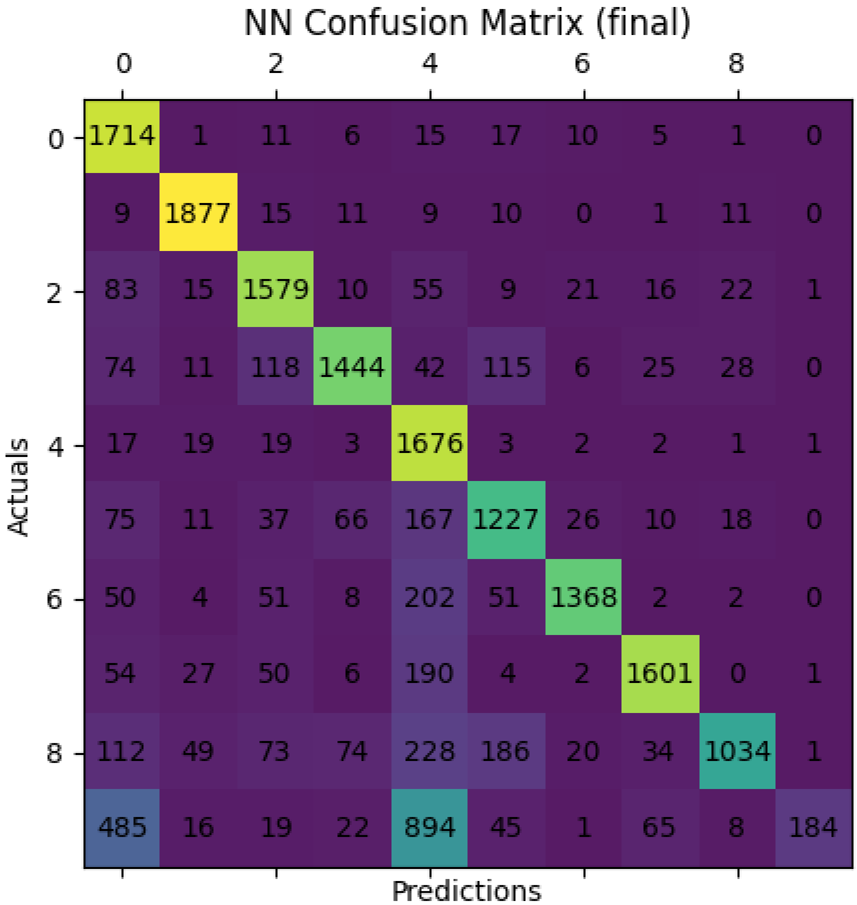
COMP 5630-001

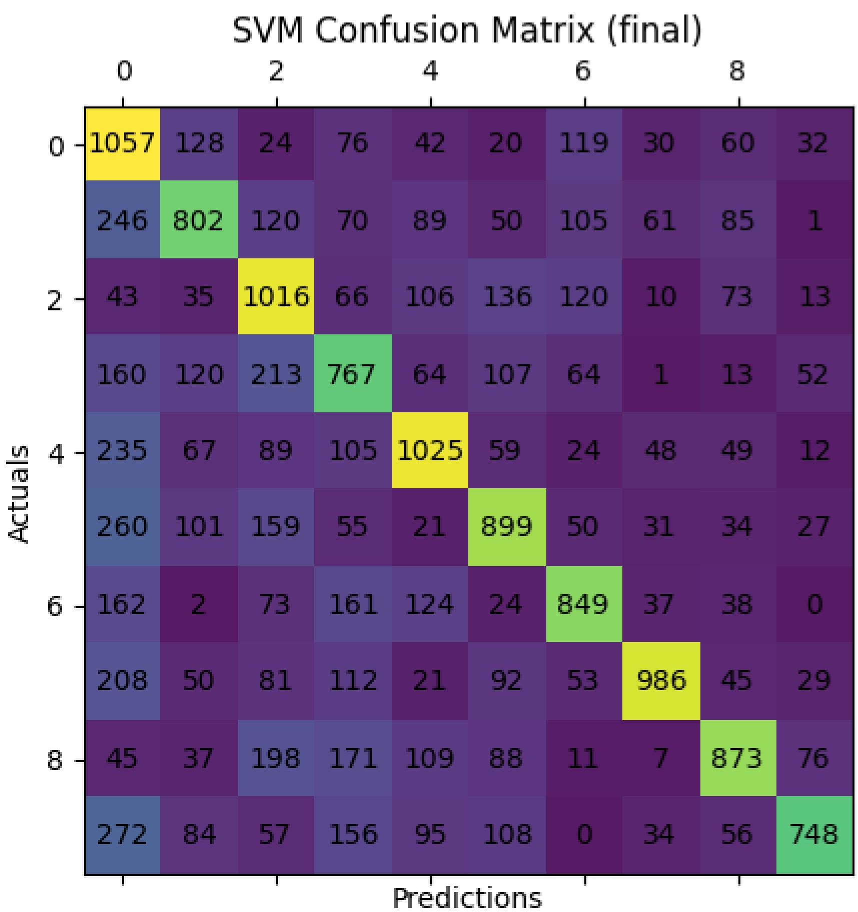
Mini-Project 2

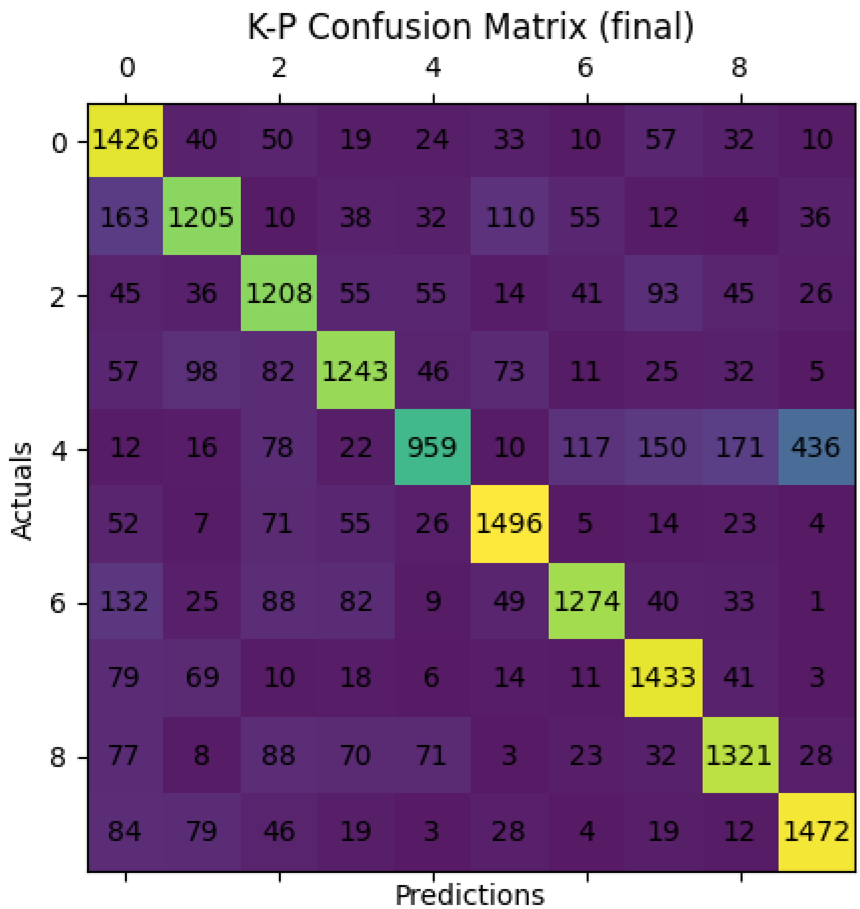
Included are train vs test accuracies for each method, along with a confusion matrix against the test data. Below that is code from the relevant files for the dataset handler, final neural network, and SVM/kernel perceptron with ECOC.

- Perceptron performed the best at 0.914/0.881 train/test accuracy after a lot of tweaking. It's biggest opportunity from the confusion matrix seems to be classifying 4's, which it confuses with 9's relatively frequently. This is understandable due to the similarities between the digits.
- Neural network was the runner up at 0.829/0.812 train/test accuracy. A large portion of the inaccuracies come from misclassifying 9's. The network chooses a 4, or to a lesser extent a 0, more often than it correctly classifies 9's. It is curious that we do not see similar misclassifications of the 4's, but perhaps the model ran away with leaning towards outputting a 4 when in doubt, as digits 5 through 8 also see a relatively high level of misclassification as a 4.
- Finally, SVM struggled the most at 0.761/0.631 train/test accuracies. I poked and prodded, but could not achieve a better performance as some absurd fit times were slowing me down. The confusion matrix doesn't give much insight, besides that the method seems to fall back on outputting 0 or 2 more than anything else when in doubt.
- Had difficulty finding a happy medium of iterations throughout. Especially in the neural network, it was a fine line between not enough iterations to learn meaningful information and so many that the model predicted the training data at 100% but failed to generalize well. I'm thinking this may be a large opportunity for the SVM method, as it was difficult to run trials with the fit times as they were.




NN Confusion Matrix (final)
0 2 4 6 8
Actuals
Predictions
1714 1 11 6 15 17 10 5 1 0
9 1877 15 11 9 10 0 1 11 0
83 15 1579 10 55 9 21 16 22 1
74 11 118 1444 42 115 6 25 28 0
17 19 19 3 1676 3 2 2 1 1
75 11 37 66 167 1227 26 10 18 0
50 4 51 8 202 51 1368 2 2 0
54 27 50 6 190 4 2 1601 0 1
112 49 73 74 228 186 20 34 1034 1
485 16 19 22 894 45 1 65 8 184


SVM Confusion Matrix (final)
0 2 4 6 8
Actuals
Predictions
1057 128 24 76 42 20 119 30 60 32
246 802 120 70 89 50 105 61 85 1
43 35 1016 66 106 136 120 10 73 13
160 120 213 767 64 107 64 1 13 52
235 67 89 105 1025 59 24 48 49 12
260 101 159 55 21 899 50 31 34 27
162 2 73 161 124 24 849 37 38 0
208 50 81 112 21 92 53 986 45 29
45 37 198 171 109 88 11 7 873 76
272 84 57 156 95 108 0 34 56 748




```
import numpy as np
from sklearn.datasets import load_svmlight_file
import pickle

class DataSet(object):
    def __init__(self):

        self.data, self.labels = self.readPickles()
        self.testData, self.testLabels = [], []
        self.trainData, self.trainLabels = [], []
        self.inputSize = len(self.data[0])
        self.hiddenSize = 100
        self.outputSize = 10

        self.hiddenValues = []
        self.outputValues = []
        self.inputWeights = []
        self.outputWeights = []
        self.biases = np.random.rand(2,1)

    # must have initialized dataset with openData()
    def readPickles(self):
        with open('data.scale', 'rb') as f:
            data = pickle.load(f)
        f.close()
```

```
        with open('label.scale', 'rb') as f:
            label = pickle.load(f)
        f.close()
        return data, label

    def openData(self):
        x, label = load_svmlight_file('MnistORIG.scale')
        x= x.toarray()
        label = label.astype(int)
        return x, label

    def setLabels(self):
        oldLabels = self.labels
        self.labels = np.zeros((len(oldLabels), 10))
        for i in range(len(oldLabels)):
            index = oldLabels[i]
            self.labels[i][index] = 1

    def splitData(self, splitAt):
        self.testData, self.testLabels = self.data[splitAt:],
self.labels[splitAt:]
        self.trainData, self.trainLabels = self.data[:splitAt],
self.labels[:splitAt]

import numpy as np
import operator
import time
from manageDataset import DataSet
from sklearn.metrics import confusion_matrix
import pickle


def sig(x, deriv=False):
    if (deriv==True):
        return sig(x) * (1 - sig(x))
    return 1/(1+np.exp(-x))

def train():
    startTime = time.time()
    X = log.trainData
    y = log.trainLabels

    np.random.seed(1)

    web1 = 2 * np.random.random((780,100)) - 1
    web2 = 2 * np.random.random((100,10)) - 1
    bias = 2 * np.random.random((2,1)) - 1
    np.seterr(all='ignore')
    for i in range(1000):
        inputChain = X
        hiddenChain = sig(np.dot(inputChain, web1) + bias[0][0])
        outputChain = sig(np.dot(hiddenChain, web2) + bias[1][0])

        outputError = y - outputChain
        if (i%100) == 0:
            print(f"Iteration {i} Error: {float(np.mean(np.abs(outputError)))}")
```

```
        outputChange = outputError * sig(outputChain, deriv=True)
        hiddenError = outputChange.dot(web2.T)
        hiddenChange = hiddenError * sig(hiddenChain, deriv=True)
        biasChange1 = np.mean(hiddenError) * sig(bias[0][0], deriv=True)
        biasChange2 = np.mean(outputError) * sig(bias[1][0], deriv=True)

        web2 += hiddenChain.T.dot(outputChange)
        web1 += inputChain.T.dot(hiddenChange)
        bias[0][0] *= biasChange1
        bias[1][0] *= biasChange2
    log.inputWeights = web1
    log.outputWeights = web2
    print(f"Finished Training, took {time.time() - startTime} seconds")

def predict(predData, trueLabels):
    i = 0
    correct = 0
    realLabels = []
    predLabels = []
    for item in predData:
        log.hiddenValues = sig(np.dot(item, log.inputWeights))
        log.outputValues = sig(np.dot(log.hiddenValues, log.outputWeights))
        index, value = max(enumerate(log.outputValues),
key=operator.itemgetter(1))
        indexTrue, valueTrue = max(enumerate(trueLabels[i]),
key=operator.itemgetter(1))
        predLabels.append(index)
        realLabels.append(indexTrue)
        #print(f"Prediction: {index} vs Actual: {indexTrue}")
        if index == indexTrue:
            correct += 1
        i += 1
    print(f"Accuracy: {correct/len(predData)}")
    confMatrix = confusion_matrix(realLabels, predLabels)
    print(confMatrix)
    with open('conf.txt', 'wb') as f:
        pickle.dump(confMatrix, f)
    f.close()

if __name__ == "__main__":
    log = DataSet()
    log.setLabels()
    log.splitData(42000)
    print("Training...")
    train()
    print("Accuracy against Training Data")
    predict(log.trainData, log.trainLabels)
    print("Accuracy against Test Data")
    predict(log.testData, log.testLabels)


import numpy as np
from manageDataset import DataSet


def binaryTestTrim(log):
    xVals = []
```

```python
    yVals = []
    for i in range(len(log.labels)):
        if log.labels[i] == 0:
            yVals.append(-1)
            xVals.append(log.data[i])
        elif log.labels[i] == 1:
            yVals.append(1)
            xVals.append(log.data[i])
        else:
            pass
    log.data = xVals
    log.labels = yVals


def linearSVM(log, iters):
    weights = pegSVM(log.trainData, log.trainLabels,iters)
    errors = 0
    for i in range(len(log.testLabels)):
        decision = weights @ log.testData[i].T
        if decision < 0:
            prediction = -1
        else:
            prediction = 1
        if prediction != log.testLabels[i]: errors += 1
    return 1 - errors/len(log.testLabels)

def kernFunc(x, y):
    mean = np.linalg.norm(x - y)**2
    variance = 1
    return np.exp(-mean/(2*variance))

def multiSVM(log, iters):
    weights = pegPercKernel(log.trainData, log.trainLabels, kernFunc, iters)
    errors = 0
    for i in range(len(log.testLabels)):
        decision = 0
        for j in range(len(log.testLabels)):
            decision += weights[j] * log.trainLabels[j] *
kernFunc(log.trainData[j], log.testData[i])
        if decision < 0:
            prediction = -1
        else:
            prediction = 1
        if prediction != log.testLabels[i]: errors += 1
    return 1 - errors/len(log.testLabels)

def pegSVM(x, y, iterations, lam=0.1):
    weights = np.zeros(x[0].shape)
    for i in range(iterations):
        iterCount = randint(0, len(y)-1)
        step = 1/(lam*(i+1))
        decision = y[iterCount] * weights @ x[iterCount].T
        if decision < 1:
            weights = (1 - step*lam) * weights + step*y[iterCount]*x[iterCount]
        else:
            weights = (1 - step*lam) * weights
    return weights
```

```
def pegPercKernel(x, y, kernel, iterations, lam=0.1):
    weights = np.zeros(len(y))
    for _ in range(iterations):
        print(f"Peg Iteration: {_}")
        it = randint(0, len(y)-1)
        decision = 0
        for j in range(len(y)):
            decision += weights[j] * y[it] * kernel(x[it], x[j])
        decision *= y[it]/lam
        if decision < 1:
            weights[it] += 1
    return weights

def testBase():
    log = DataSet()
    binaryTestTrim(log)
    log.splitData(42000)
    iters = 1000
    kernel = False

    accuracy = multiSVM(log, iters)
    print('Accuracy Against Test:', accuracy)

def sepData(log, id):
    _log = {'data': [],'labels': []}

    for i in range(len(log.trainLabels)):
        _log['data'].append(log.trainData[i])
        if log.trainLabels[i] == id:
            _log['labels'].append(1)
        else:
            _log['labels'].append(-1)
    _log['data'] = np.array(_log['data'])
    _log['labels'] = np.array(_log['labels'])
    return _log

def testECOC():
    log = DataSet()
    log.splitData(7000)
    iters = 1000
    weightClasses = []
    for i in range(log.outputSize):
        print("using RBf")
        _log = sepData(log, i)
        weightClasses.append(pegPercKernel(_log['data'], _log['labels'],
kernFunc, iters))
    errors = 0
    for i in range(len(log.testLabels)):
        predictions = []
        for k in range(10):
            weights = weightClasses[k]
            decision = 0
            for j in range(len(log.trainLabels)):
                decision += weights[j] * log.trainLabels[j] *
kernFunc(log.trainData[j], log.testData[i])
            predictions.append(decision)
```

```
        predictions = np.array(predictions)
        classLabels = predictions.argmax()
        if classLabels != log.testLabels[i]:
            errors += 1
            print(f"Error: {classLabels} found, {log.testLabels[i]} expected,
index {i}")
    accuracy = 1 - errors / len(log.testLabels)
    print(f"Error: {errors / len(log.testLabels)}")
    print(f"Accuracy: {accuracy}")

if __name__ == "__main__":
    testBase()
    testECOC()

import numpy as np
from manageDataset import DataSet

class PolynomialPerceptron():
    def __init__(self, inputs, targets, n, p, max_iter):
        self.inputs = inputs
        self.targets = targets
        self.p = p
        self.n = n
        self.max_iter = max_iter
        self.alpha = np.zeros(n)

    def polyFunc(self, X, Y, p):
        Y = Y.T
        res = np.dot(X,Y)
        res = np.add(res, Y)
        res = np.exp(res, p)
        return res

    def train(self):
        K = self.polyFunc(self.inputs, self.inputs, self.p)
        for i in range(self.max_iter):
            print(f"Iteration {i}")
            for j in range(self.n):
                alph = self.alpha * self.targets
                k = alph * K[j]
                if self.targets[j] * k <= 0:
                    self.alpha[j] += 1

    def predict(self, inputData):
        tmp = (self.alpha * self.targets) * inputData
        testPred = 1 if tmp > 0 else 0
        return testPred

    def acc(self, inputs, targets):
        correct = 0
        kVal = self.polyFunc(inputs, self.inputs, self.p)
        for idx, each in enumerate(inputs):
            correct += self.predict(kVal[idx]) == targets[idx]
        return correct / len(inputs)

log = DataSet()
log.splitData(42000)
```

```python
iters = 1000
n = 42000
c = 0.01

model = PolynomialPerceptron(log.trainData, log.trainLabels,n,c,iters)
model.train()
acc = model.acc(log.testData, log.testLabels)
print(f"accuracy: {acc}")
confMat = confusion_matrix(log.realLabels, log.testLabels)
with open('percConf.txt', 'wb') as f:
    pickle.dump(confMat, f)
f.close()
```