CS446: Machine Learning **Fall 2015**

Problem Set 2

Dominik Durner *Handed In: September 29, 2015*

Question 1

1.a

For this task I wrote a Haskell program (Figure 1) that can calculate the entropy of each variable assignment and the total entropy, as well as the gain of each variable.

```haskell
module FindRoot where

entropy positives negatives = - positvePortion * (logBase 2 positvePortion)
    - negativePortion * logBase 2 negativePortion
  where
    n = positives+negatives
    positvePortion = positives / n
    negativePortion = negatives / n

gain entropy val1 val1Entropy val2 val2Entropy = entropy
    - (val1Portion * val1Entropy + val2Portion * val2Entropy)
  where
    n = val1+val2
    val1Portion = val1 / n
    val2Portion = val2 / n
```

Figure 1: FindRoot.hs

The different entropies are given with following values.

$$H_{total} = -\frac{35}{50}ld(\frac{35}{50}) - \frac{15}{50}ld(\frac{15}{50}) = 0.8812908992306927 \rightarrow \text{ghci: entropy 35 15} \quad (1)$$

$$H_{examYes} = -\frac{15}{16}ld(\frac{15}{16}) - \frac{1}{16}ld(\frac{1}{16}) = 0.3372900666170139 \rightarrow \text{ghci: entropy 15 1} \quad (2)$$

$$H_{examNo} = -\frac{20}{34}ld(\frac{20}{34}) - \frac{14}{34}ld(\frac{14}{34}) = 0.9774178175281716 \rightarrow \text{ghci: entropy 20 14} \quad (3)$$

$$H_{holidayYes} = -\frac{5}{15}ld(\frac{5}{15}) - \frac{10}{15}ld(\frac{10}{15}) = 0.9182958340544896 \rightarrow \text{ghci: entropy 5 10} \quad (4)$$

$$H_{holidayNo} = -\frac{30}{35}ld(\frac{30}{35}) - \frac{5}{35}ld(\frac{5}{35}) = 0.5916727785823275 \rightarrow \text{ghci: entropy 30 5} \quad (5)$$

Afterwards, the gain can be calculated to determine if Exam or Holiday should be considered for the root label.

$$G_{exam} = H_{total} - (\frac{16}{50}H_{examYes} + \frac{34}{50}H_{examNo}) = 0.10871396199409156 \tag{6}$$

$$\rightarrow \text{ghci: gain (entropy 35 15) 16 (entropy 15 1) 34 (entropy 20 14)} \tag{7}$$

$$G_{holiday} = H_{total} - (\frac{15}{50}H_{holidayYes} + \frac{35}{50}H_{holidayNo}) = 0.19163120400671674 \tag{8}$$

$$\rightarrow \text{ghci: gain (entropy 35 15) 15 (entropy 5 10) 35 (entropy 30 5)} \tag{9}$$

Therefore, the splitting should be rooted with the option if there is holiday or not.

1.b

The algorithm denoted to minimize the majorityError can be found in Figure 2. If more values had the same error I just chose randomly one.

```
if colour = purple
  if act = dip
    class = F
  if act != dip
    if age = child
      class = F
    if age != child
      class = T
if colour != purple
  if size = small
    class = T
  if size != small
    if age = child
      class = F
    if age != child
      if act = dip
        class = F
      if act != dip
        class = T
```

Figure 2: majorityError.algorithm

1.c

No! Since finding the minimal/optimal decision tree is NP hard, the ID3 algorithm does not guarantee to give me the best possible one. ID3 is just using a greedy strategy to find the next best splitting node. Accordingly, the ID3 algorithm with information gain as heuristic tries to optimize the tree locally. Hence, it tries to break the tree at the variable with the most valuable information right now. But it might be the case that splitting on a less important variable helps to reduce the decision tree afterwards. As a conclusion we can say that ID3 finds a pretty good tree but it doesn't need to be optimal!

Question 2

Feature Generation

Beside the requested 260 Features firstNamei=k and LastNamei=k, $i \in 1..5, k \in a, .., z$, I implemented more features. The feature firstNameLonger looks whether the firstname or lastname has more characters. firstNameLengthEven and lastNameLengthEven checks whether the string consists of an even amount of characters. These features just increased the accuracy a little bit so I went on for more features. As described in 2a, you suggested to maybe look at combinations and contains. Therefore I implemented the second one. I used 3*26 features which include letterInName=k, letterInFirstName=k, letterInLastName=k, $k \in a, .., z$. These feature look if one letter is just existing in the string or not. Since I got a huge increase of accuracy for all algorithms using these features, I think that those features might be in connection to the real function.

Comparison

Table 1: Comparison of the five different learning approaches

Algorithm	Parameters	Accuracy	99% Interval	Significance
SGD with ID3-4	$E = 0.1, R = 0.005$	$p_A = 85.37\%$[1]	[74.94%, 95.80%]	$2.2363 \Rightarrow No!$
SGD	$E = 0.075, R = 0.0005$	$p_A = 81.29\%$[1]	[70.53%, 92.05%]	$0.1062 \Rightarrow No!$
ID3-4 stumps	-	$p_A = 80.91\%$	[60.86%, 100%]	$1.1700 \Rightarrow No!$
ID3-8 stumps	-	$p_A = 78.88\%$	[57.60%, 100%]	No Difference!
ID3 full tree	-	$p_A = 78.88\%$	[57.60%, 100%]	-

Table 2: Accuracies for different folds

Algorithm	fold1	fold2	fold3	fold4	fold5	p_A	σ
SGD with ID3-4	86.44%	93.22%	84.75%	79.66%	82.76%	85.37%	0.050638
SGD	77.97%	86.44%	86.44%	74.58%	81.03%	81.29%	0.052240
ID3-4 stumps	77.97%	93.22%	86.44%	79.67%	67.24%	80.91%	0.097361
ID3-8 stumps	69.49%	93.22%	84.75%	77.97%	68.97%	78.88%	0.103356
ID3 full tree	69.49%	93.22%	84.75%	77.97%	68.97%	78.88%	0.103356

For the calculation we used following equations.

$$p_A = \frac{1}{n} \sum_{i=1}^{n} a_i \tag{10}$$

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (a_i - p_A)^2} \tag{11}$$

[1]Both SGD algorithms are due to stochastic inaccuracies since I only train on random samples

To calculate the confidence interval we need to calculate the following interval.

$$99\% \text{ intervall for df} = \text{k: } [p_A - \frac{t * \sigma}{\sqrt{N}}, p_A + \frac{t * \sigma}{\sqrt{N}}], \text{ with } t = t_{0.995,k} \tag{12}$$

Since we have 5 different folds we have a degree of freedom of 4. Hence, we get for our 99% interval $t_{0.995,4} = 4.604$. Therefore we get following intervals.

$$I_{SGDStumps} = [0.8537 - \frac{4.604 * 0.050638}{\sqrt{5}}, 0.8537 + \frac{4.604 * 0.050638}{\sqrt{5}}] = [74.94\%, 95.80\%]$$
$$I_{SGD} = [0.8129 - \frac{4.604 * 0.052240}{\sqrt{5}}, 0.8129 + \frac{4.604 * 0.052240}{\sqrt{5}}] = [70.53\%, 92.05\%]$$
$$I_{ID3-4} = [0.8091 - \frac{4.604 * 0.097361}{\sqrt{5}}, 0.8091 + \frac{4.604 * 0.097361}{\sqrt{5}}] = [60.86\%, 100.95\%]$$
$$I_{ID3-8} = [0.7888 - \frac{4.604 * 0.103356}{\sqrt{5}}, 0.7888 + \frac{4.604 * 0.103356}{\sqrt{5}}] = [57.60\%, 100.16\%]$$
$$I_{ID3-inf} = [0.7888 - \frac{4.604 * 0.103356}{\sqrt{5}}, 0.7888 + \frac{4.604 * 0.103356}{\sqrt{5}}] = [57.60\%, 100.16\%]$$

To show if there is a difference or not we need to find an interval in which we accept the null hypothesis (there is no difference). The value of t is calculated with the following equation.

$$p_D = \frac{1}{n}\sum_{i=1}^{n} a_i - b_i \tag{13}$$

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}((a_i - b_i) - p_D)^2} \tag{14}$$

$$t_{N-1} = \frac{p_D * \sqrt{N}}{\sigma} \tag{15}$$

As long as the t value is in the interval $(-t_{a/2,N-1}, t_{a/2,N-1})$ we accept it. For the values given for a and the 5 folds we get the interval of $(-4.604, 4.604)$. As long as the t value for the paired test is in between this interval we don't see any statistical significance. The results can be found in the main comparison table, in which the t value of the ith row determines the t value between the ith and i+1th algorithm.

Table 3: Accuracies difference table for connected pairs of algorithms

Algorithm	fold1	fold2	fold3	fold4	fold5	p_D	σ
SGD & stumps vs. SGD	0.0847	0.0678	-0.0169	0.0508	0.0173	0.04074	0.040736
SGD vs. ID3-4 stumps	0	-0.0678	0	-0.0509	0.1379	0.00384	0.080824
ID3-4 vs. ID3-8 stumps	0.0848	0	0.0169	0.017	-0.0173	0.02028	0.038757
ID3-8 vs. ID3 full tree	0	0	0	0	0	0	0

Conclusion

First of all the combination of SGD and Decision Tree Stumps is working best but not statistical significantly best. The non significantly better values might be due to the randomness of the trees and the already good accuracy of ID3-4 stumps algorithm. If the decision stumps aren't accurate at all the SGD with stumps would have a hard time to perform well. Due to the good incoming data, the stepsize aka learning rate R can be reduced in comparison to the normal SGD. (see SGD Parameter Tuning for my tuning concept) This helps to reduce the computation time of the building of the classifier. Since my step size is quite small in the normal SGD and I really push down the error rate, the algorithm to build the SGD classifier is really slow in comparison to the easy to calculate decision trees. Since my added features describe the tree quite well, the ID3 without restrictions and the ID3 with depth of 8 have the same trees for the given folds. Hence, the accuracy and performance is the same. Since the ID3-4 is performing a little better, I think that the other ID3 algorithms already overfit the data. Overall we can see that it might be worth for this data to perform the slightly more work to build our SGD with decision stumps. This is enhanced by the results we got for the confidence interval since the deviation is way worse by ID3 than for SGD & ID3 stumps. On the other hand I get acceptable p_A (which isn't significantly worse) with the decision stumps of ID3-4 and need less computation time.

SGD Parameter Tuning

I tuned the parameters of SGD to get a better p_A accuracy. Therefore I first tried the algorithm with some combinations of errorThreshold E and learningRate R to get a feeling for the size ranges which perform best. After having a better understanding which numbers might be suitable for the algorithms (E in the range of $[10^{-2}, 10^{0}]$ and R in the range of $[10^{-4}, 10^{-2}]$), I implemented an testing procedure. Therefore, I implemented loops for both variables and tested it several times in a binary search fashion attempt for the variable values (getting more specific on the best found ones) and calculated for each of these iterations their performance. In the end I took the values that performed best.

SGD Implementation

```java
    public void learnWeightVector(double threshold, double[][] x, double[] t,
        double R) {
        generateStartWeightVector(x[0].length);
        double error;
        do {
            double[] res = new double[w.length];
            int randomDataSet = (int) (Math.random() * x.length);
            for (int i = 0; i < w.length; i++) {
                res[i] = R * (t[randomDataSet] - dotProduct(w, x[randomDataSet
                    ])) * x[randomDataSet][i];
            }
            for (int i = 0; i < w.length; i++) {
                w[i] += res[i];
            }
            error = currentError(x, t);
            R = updateLearningRate(R);
        } while (threshold <= error);
    }

    private double currentError(double[][] x, double[] t) {
        double error = 0.0;
        for (int i = 0; i < x.length; i++) {
            error += Math.pow(t[i] - dotProduct(w, x[i]), 2);
        }
        return (0.5*error)/((double) x.length);
    }
```

Figure 3: SGD weight updates

In Figure 3 the main methods for the Stochastic Gradient Descent are shown. It learns weights as long as the currentError is greater than a predefined threshold. In each iteration the algorithm chooses an arbitrary element of the training data set and updates the weight vector accordingly to the classification of the chosen data point. The error function is averaged on the number of samples to have an error that isn't automatically growing with more test data (as discussed in an office hour). Therefore the error function is the following.

$$E = \frac{1}{2 * |D|} \sum_{d \in D} (t_d - o_d)^2 \tag{16}$$

```java
    @Override
    public void buildClassifier(Instances instances) throws Exception {
        instances = new Instances(instances);
        instances.deleteWithMissingClass();

        int numAttributes = instances.firstInstance().numAttributes();
        double[][] x = new double[instances.numInstances()][numAttributes -
            1];
        double[] t = new double[instances.numInstances()];

        for (int currentInstance = 0; currentInstance < instances.numInstances
            (); currentInstance++) {
            for (int currentAttribute = 0; currentAttribute < numAttributes -
                1; currentAttribute++) {
                x[currentInstance][currentAttribute] = instances.instance(
                    currentInstance).value(currentAttribute) == 0 ? -1 : 1;
            }

            t[currentInstance] = instances.instance(currentInstance).
                classValue() == 0 ? -1 : 1;
        }
        learnWeightVector(errorThreshold, x, t, learningRate);
    }

    @Override
    public double classifyInstance(Instance instance) throws java.lang.
        Exception {
        int numAttributes = instance.numAttributes();
        double[] x = new double[numAttributes - 1];
        for (int currentAttribute = 0; currentAttribute < numAttributes - 1;
            currentAttribute++) {
            x[currentAttribute] = instance.value(currentAttribute) == 0 ? -1 :
                1;
        }
        return dotProduct(w, x) >= 0 ? 1 : 0;
    }
```

Figure 4: Weka Instances to SGD

Since the Weka Format gives us 0 and 1 as the possible feature values, I translate those values to the standard -1 and 1 notation. Therefore, I needed to implement the methods buildClassifier and classifyInstance to use my SGD algorithm with the Weka Framework. Every variable with 0 is mapped to -1 and every 1 is mapped to 1. The classification new instances with the SGD is the signum of the dot product between the weight vector and the instance that needs to be classified. Therefore I also have to remap those positive and negative values to the 0 and 1 notation afterwards, to classify the instances correctly within the Weka Framework. (see Figure 4)

SGD with Decision Stumps Implementation

For this problem I solved it by having first all combinations of train and test data. For each of the combinations I am creating 100 decision trees with a half of the samples in the train data (I shuffle the data and then I delete half of it for each tree). After having the 100 decision trees, I am generating the results of their answers for both the training samples and the test set and label this 100 dimensional vector with the real label of the corresponding badge. The generated train set is now used to build my SGD classifier. Afterwards, I am testing my SGD with the test data set and calculate the accuracy. As stated at the beginning I am doing this for all test and train data combinations to do the five fold validation afterwards.

```java
public static void main(String[] args)throws Exception{
  //Five Fold:
  int acc=0;
  for(int i=1;i<6;i++){
    Id3[] trees=new Id3[100];
    Instances test=null,train=null;
    for(int j=1;j<6;j++){
      if(i==j)
        test=FeatureGenerator.readData(args[0]+".fold"+i);
      else{
        if(train!=null){
          Instances is=FeatureGenerator.readData(args[0]+".fold"+j);
          for(int instInd=0;instInd<is.numInstances();instInd++)
            train.add(is.instance(instInd));
        }
        else train=FeatureGenerator.readData(args[0]+".fold"+j);
      }
    }
    //build the 100 decision trees from random 50% of the train data
    initializeDecisionTreeStumps(trees,train);
    // build the dataset from the decision trees
    Instances sgdTrain=calculateSGDInstances(trees,train);
    Instances sgdTest=calculateSGDInstances(trees,test);
    StochasticGradientDescent sgd=new StochasticGradientDescent();
    sgd.errorThreshold=25;sgd.learningRate=0.005;
    sgd.buildClassifier(sgdTrain);
    Evaluation evaluation=new Evaluation(sgdTest);
    evaluation.evaluateModel(sgd,sgdTest);
    acc+=evaluation.pctCorrect();
  }
  System.out.println("SGD with ID3-4 stubs acc:"+((double)acc)/5);
}

private static void initializeDecisionTreeStumps(Id3[] trees,Instances train)
    throws Exception{
  for(int j=0;j<trees.length;j++){
    Id3 classifier=new Id3();
    Instances t=new Instances(train);
    t.randomize(new Random());
    int n=t.numInstances();
    for(int r=n-1;r>=n/2;r--){
      t.delete(r);
    }
    classifier.setMaxDepth(4);
    classifier.buildClassifier(t);
    trees[j]=classifier;
    }
  }
}
```

Figure 5: Calculating the ID3-4 stumps and using them for SGD

Decision Trees

Now I want to present you for each of the three ID3 algorithms the best decision tress. All of the trees were found on the second fold and had for all three trees an accuracy of 93.22%. For the ID3-8 and ID3-inf the tree is the same, since the infinity version also only needs 5 levels to classify the data. The data and the evaluation summary can be also found in the source code zip.

```
letterInFirstName=a = 1
|   letterInLastName=t = 1
|   |   lastNameLengthEven = 1
|   |   |   letterInLastName=e = 1: +
|   |   |   letterInLastName=e = 0
|   |   |   |   lastName4=u = 1: -
|   |   |   |   lastName4=u = 0: +
|   |   lastNameLengthEven = 0: +
|   letterInLastName=t = 0
|   |   letterInLastName=r = 1
|   |   |   lastName4=e = 1
|   |   |   |   letterInLastName=a = 1: +
|   |   |   |   letterInLastName=a = 0: -
|   |   |   lastName4=e = 0
|   |   |   |   firstName2=i = 1: -
|   |   |   |   firstName2=i = 0: +
|   |   letterInLastName=r = 0
|   |   |   letterInLastName=h = 1
|   |   |   |   firstName4=i = 1: -
|   |   |   |   firstName4=i = 0: +
|   |   |   letterInLastName=h = 0
|   |   |   |   letterInLastName=o = 1: +
|   |   |   |   letterInLastName=o = 0: -
letterInFirstName=a = 0
|   letterInFirstName=n = 1
|   |   letterInName=s = 1
|   |   |   firstName1=o = 1
|   |   |   |   lastName0=c = 1: -
|   |   |   |   lastName0=c = 0: +
|   |   |   firstName1=o = 0: -
|   |   letterInName=s = 0
|   |   |   letterInLastName=h = 1: +
|   |   |   letterInLastName=h = 0
|   |   |   |   letterInName=r = 1: +
|   |   |   |   letterInName=r = 0: -
|   letterInFirstName=n = 0
|   |   letterInFirstName=d = 1
|   |   |   lastName0=p = 1: -
|   |   |   lastName0=p = 0: +
|   |   letterInFirstName=d = 0: -
```

Figure 6: ID3-4 tree on fold 2

```
letterInFirstName=a = 1
|   letterInLastName=t = 1
|   |   lastNameLengthEven = 1
|   |   |   letterInLastName=e = 1: +
|   |   |   letterInLastName=e = 0
|   |   |   |   lastName4=u = 1: −
|   |   |   |   lastName4=u = 0
|   |   |   |   |   firstName1=r = 1: −
|   |   |   |   |   firstName1=r = 0: +
|   |   lastNameLengthEven = 0: +
|   letterInLastName=t = 0
|   |   letterInLastName=r = 1
|   |   |   lastName4=e = 1
|   |   |   |   letterInLastName=a = 1
|   |   |   |   |   firstName0=t = 1: −
|   |   |   |   |   firstName0=t = 0: +
|   |   |   |   letterInLastName=a = 0
|   |   |   |   |   lastNameLengthEven = 1: −
|   |   |   |   |   lastNameLengthEven = 0: +
|   |   |   lastName4=e = 0
|   |   |   |   firstName2=i = 1: −
|   |   |   |   firstName2=i = 0
|   |   |   |   |   lastName4=a = 1
|   |   |   |   |   |   letterInName=h = 1: +
|   |   |   |   |   |   letterInName=h = 0
|   |   |   |   |   |   |   firstName0=p = 1: +
|   |   |   |   |   |   |   firstName0=p = 0: −
|   |   |   |   |   lastName4=a = 0: +
|   |   letterInLastName=r = 0
|   |   |   letterInLastName=h = 1
|   |   |   |   firstName4=i = 1: −
|   |   |   |   firstName4=i = 0: +
|   |   |   letterInLastName=h = 0
|   |   |   |   letterInLastName=o = 1
|   |   |   |   |   lastName1=a = 1
|   |   |   |   |   |   firstName0=n = 1: +
|   |   |   |   |   |   firstName0=n = 0: −
|   |   |   |   |   lastName1=a = 0: +
|   |   |   |   letterInLastName=o = 0: −
letterInFirstName=a = 0
|   letterInFirstName=n = 1
|   |   letterInName=s = 1
|   |   |   firstName1=o = 1
|   |   |   |   lastName0=c = 1: −
|   |   |   |   lastName0=c = 0: +
|   |   |   firstName1=o = 0: −
|   |   letterInName=s = 0
|   |   |   letterInLastName=h = 1: +
|   |   |   letterInLastName=h = 0
|   |   |   |   letterInName=r = 1
|   |   |   |   |   firstName2=r = 1: −
|   |   |   |   |   firstName2=r = 0: +
|   |   |   |   letterInName=r = 0: −
|   letterInFirstName=n = 0
|   |   letterInFirstName=d = 1
|   |   |   lastName0=p = 1: −
|   |   |   lastName0=p = 0: +
|   |   letterInFirstName=d = 0: −
```

Figure 7: ID3-8 tree on fold 2

```
letterInFirstName=a = 1
|   letterInLastName=t = 1
|   |   lastNameLengthEven = 1
|   |   |   letterInLastName=e = 1: +
|   |   |   letterInLastName=e = 0
|   |   |   |   lastName4=u = 1: −
|   |   |   |   lastName4=u = 0
|   |   |   |   |   firstName1=r = 1: −
|   |   |   |   |   firstName1=r = 0: +
|   |   lastNameLengthEven = 0: +
|   letterInLastName=t = 0
|   |   letterInLastName=r = 1
|   |   |   lastName4=e = 1
|   |   |   |   letterInLastName=a = 1
|   |   |   |   |   firstName0=t = 1: −
|   |   |   |   |   firstName0=t = 0: +
|   |   |   |   letterInLastName=a = 0
|   |   |   |   |   lastNameLengthEven = 1: −
|   |   |   |   |   lastNameLengthEven = 0: +
|   |   |   lastName4=e = 0
|   |   |   |   firstName2=i = 1: −
|   |   |   |   firstName2=i = 0
|   |   |   |   |   lastName4=a = 1
|   |   |   |   |   |   letterInName=h = 1: +
|   |   |   |   |   |   letterInName=h = 0
|   |   |   |   |   |   |   firstName0=p = 1: +
|   |   |   |   |   |   |   firstName0=p = 0: −
|   |   |   |   |   lastName4=a = 0: +
|   |   letterInLastName=r = 0
|   |   |   letterInLastName=h = 1
|   |   |   |   firstName4=i = 1: −
|   |   |   |   firstName4=i = 0: +
|   |   |   letterInLastName=h = 0
|   |   |   |   letterInLastName=o = 1
|   |   |   |   |   lastName1=a = 1
|   |   |   |   |   |   firstName0=n = 1: +
|   |   |   |   |   |   firstName0=n = 0: −
|   |   |   |   |   lastName1=a = 0: +
|   |   |   |   letterInLastName=o = 0: −
letterInFirstName=a = 0
|   letterInFirstName=n = 1
|   |   letterInName=s = 1
|   |   |   firstName1=o = 1
|   |   |   |   lastName0=c = 1: −
|   |   |   |   lastName0=c = 0: +
|   |   |   firstName1=o = 0: −
|   |   letterInName=s = 0
|   |   |   letterInLastName=h = 1: +
|   |   |   letterInLastName=h = 0
|   |   |   |   letterInName=r = 1
|   |   |   |   |   firstName2=r = 1: −
|   |   |   |   |   firstName2=r = 0: +
|   |   |   |   letterInName=r = 0: −
|   letterInFirstName=n = 0
|   |   letterInFirstName=d = 1
|   |   |   lastName0=p = 1: −
|   |   |   lastName0=p = 0: +
|   |   letterInFirstName=d = 0: −
```

Figure 8: ID3-infinity tree on fold 2