Homework 2 for CSI 431/531

Due: Thursday, Oct 19, 2:44pm (before class)

All homeworks are individual assignments. This means: write your own solutions and do not copy code/solutions from peers or online. Should academic dishonesty be detected, the proper reporting protocols will be invoked (see Syllabus for details).

Instructions: Submit two files. One should be a write-up of all solutions and observations, as *Solution.pdf*. The second should be an archive *Code.zip* containing code and any relevant results files.

1. **Decision trees** As part of this question you will implement and compare the Information Gain, Gini Index and CART evaluation measures for splits in decision tree construction.Let $D = (X, y), |D| = n$ be a dataset with n samples. The entropy of the dataset is defined as

$$H(D) = -\sum_{i=1}^{2} P(c_i|D) log_2 P(c_i|D),$$

where $P(c_i|D)$ is the fraction of samples in class i. A split on an attribute of the form $X_i \leq c$ partitions the dataset into two subsets D_Y and D_N based on whether samples satisfy the split predicate or not respectively. The split Entropy is the weighted average Entropy of the resulting datasets D_Y and D_X:

$$H(D_Y, D_N) = \frac{n_Y}{n} H(D_Y) + \frac{n_N}{n} H(D_N),$$

where n_Y are the number of samples in D_Y and n_N are the number of samples in D_N. The Information Gain (IG) of a split is defined as the the difference of the Entropy and the split entropy:

$$IG(D, D_Y, D_N) = H(D) - H(D_Y, D_N). \tag{1}$$

The **higher** the information gain the better.

The Gini index of a data set is defined as $G(D) = 1 - \sum_{i=1}^{2} P(c_i|D)^2$ and the Gini index of a split is defined as the weighted average of the Gini indices of the resulting partitions:

$$G(D_Y, D_N) = \frac{n_Y}{n} G(D_Y) + \frac{n_N}{n} G(D_N). \tag{2}$$

The **lower** the Gini index the better.

Finally, the CART measure of a split is defined as:

$$CART(D_Y, D_N) = 2\frac{n_Y}{n}\frac{n_N}{n} \sum_{i=1}^{2} |P(c_i|D_Y) - P(c_i|D_N)|. \tag{3}$$

The **higher** the CART the better.

You will need to fill in the implementation of the three measures in the provided Python code as part of the homework. **Note: You are not allowed to use existing implementations of the measures.**

The homework includes two data files, *train.txt* and *test.txt*. The first consists of 100 observations to use to train your classifiers; the second has 10 to test. Each file is comma-separated, and each row contains 11 values - the first 10 are attributes (a mix of numeric and categorical translated to numeric, e.g. {T,F} = {0,1}), and the final being the true class of that observation. You will need to separate attributes and class in your $load(filename)$ function.

You are also given *HW2.py*. This contains all necessary functions with additional details in the docstrings. Implement each function within the skeleton given. You are allowed to add any helper functions as desired. The functions provided are what will be graded, along with solutions written up.

(a) **[10 pts.]** Implement the $IG(D, index, value)$ function according to equation 1, where D is a dataset, $index$ is the index of an attribute and $value$ is the split value such that the split is of the form $X_i \leq value$. The function should return the value of the information gain.
Solution:

```
def doSplit(D, index, value):
    X = D[0]
    y = D[1]

    classes = [c for c in set(y)]
    #The below is fine for a small number of classes; for very many indices, avoid dicts
    partitioned_counts = {decision:{c:0 for c in classes} for decision in range(2)}

    for i in range(len(X)):
        decision = 0 if X[i][index]<=value else 1
        partitioned_counts[decision][y[i]]+=1

    counts = tuple([np.sum([partitioned_counts[d][c] for c in classes]) for d in range(2)])
    probs = tuple([[(float(partitioned_counts[d][c])/counts[d]) if counts[d] > 0 else 0
                    for c in classes] for d in range(2)])

    return counts, probs, classes


def IG(D, index, value):
    counts, probs, classes = doSplit(D, index, value)
    N = float(np.sum(counts))
    y = D[1].tolist()
    probs_orig = [y.count(c)/N for c in set(y)]

    HD = -2*np.sum([p*np.log2(p) for p in probs_orig])
    HDy = -2*np.sum([p*np.log2(p) for p in probs[0] if p > 0])
    HDn = -2*np.sum([p*np.log2(p) for p in probs[1] if p > 0])

    IG = HD - counts[0]/N*HDy - counts[1]/N*HDn

    return IG
```

(b) **[10 pts.]** Implement the $G(D, index, value)$ function according to equation 2, where D is a dataset, $index$ is the index of an attribute and $value$ is the split value such that the split is of the form $X_i \leq value$. The function should return the value of the gini index value.
Solution:

```
def G(D, index, value):
    counts, probs, classes = doSplit(D, index, value)
    N = float(np.sum(counts))

    GDy = 1-np.sum([p**2 for p in probs[0]])
    GDn = 1-np.sum([p**2 for p in probs[1]])

    Gini = counts[0]/N*GDy + counts[1]/N*GDn

    return Gini
```

(c) **[10 pts.]** Implement the $CART(D, index, value)$ function according to equation 3, where D is a dataset, $index$ is the index of an attribute and $value$ is the split value such that the split is of the form $X_i \leq value$. The function should return the value of the CART value.
Solution:

```
def CART(D, index, value):
    counts, probs, classes = doSplit(D, index, value)
    N = float(np.sum(counts))
```

```
    CART = 2* (counts[0]/N)* (counts[1]/N) * np.sum([abs(probs[0][i] - probs[1][i])
                for i in range(len(probs[0]))])

    return CART
```

(d) **[20 pts.]** Implement the function $bestsplit(D, criterion)$ which takes as an input a dataset D, a string value from the set $\{$"IG", "$GINI$", "$CART$"$\}$ which specifies a measure of interest. This function should return the best possible split for measure *criterion* in the form of a tuple $(i, value)$, where i is the attribute index and *value* is the split value. The function should probe all possible values for each attribute and all attributes to form splits of the form $X_i \leq value$.

Solution:

```
def bestSplit(D, criterion):
    #functions are first class objects in python, so let's refer to our desired criterion by a
    if criterion == "IG": criterion_function=IG
    if criterion == "GINI": criterion_function=G
    if criterion == "CART": criterion_function=CART

    X = D[0]
    num_attr = X.shape[1]
    split_scores = {}
    for index in range(num_attr):
        vals = sorted(set(X[:,index]))
        for value in vals:
            split_scores[(index,value)] = criterion_function(D, index, value)

    if criterion == "IG" or criterion == "CART":
        return max(split_scores, key=split_scores.get)
    else:
        return min(split_scores, key=split_scores.get)
```

(e) **[10 pts]** Load the training data "train.txt" provided in the homework by implementing the function $load(filename)$, which should return a dataset D, and find the best possible split for each of the three criteria for the data (Hint: Note that some measures need to be minimized and some maximized).

Solution:

```
def load(filename):
    all_data = np.loadtxt(filename, delimiter=",")
    X = all_data[:,:-1]
    y = all_data[:,-1]

    return (X, y)
```

(f) **[10 pts]** Assume that you built a very simple DT based on the only best split for each criterion from the previous question. Show the three decision trees resulting from the best splits (draw the trees, split points and decision nodes). Assume that in each of the two leaves of the tree a decision is made to classify as the majority class.

Solution:

```
def bestSplit(D, criterion):
    #functions are first class objects in python, so let's refer to our desired criterion by a
    if criterion == "IG": criterion_function=IG
    if criterion == "GINI": criterion_function=G
    if criterion == "CART": criterion_function=CART

    X = D[0]
    num_attr = X.shape[1]
    split_scores = {}
    for index in range(num_attr):
        vals = sorted(set(X[:,index]))
        for value in vals:
            split_scores[(index,value)] = criterion_function(D, index, value)
```

```
    if criterion == "IG" or criterion == "CART":
        return max(split_scores, key=split_scores.get)
    else:
        return min(split_scores, key=split_scores.get)
```

(g) **[30 pts]** How many classification errors will the above three classifiers make on the testing set provided in "test.txt". A classification error occurs when a testing instance is not classified as its correct class. Note, that you need to implement the three functions $classifyIG(train, test)$, $classifyG(train, test)$, $classifyCART(train, test)$ based on the best splits you found in the previous parts. Each function should take in training data, create the best single split on a single attribute using the above defined functions, and classify each observation in the test data. The output should be a list of predicted classes for the test data (in the same order, of course). Then compare these predicted classes to the true classes of the test data.

Solution:

```
def classifyIG(train, test):
    (index, value) = bestSplit(train, "IG")
    counts, probs, classes = doSplit(train, index, value)
    classifier = [np.argmax(probs[0]), np.argmax(probs[1])]


    pred = []
    for obs in test[0]:
        decision = 0 if obs[index] <= value else 1
        pred.append(classifier[decision])

    return pred

def classifyG(train, test):
    (index, value) = bestSplit(train, "GINI")
    counts, probs, classes = doSplit(train, index, value)
    classifier = [np.argmax(probs[0]), np.argmax(probs[1])]


    pred = []
    for obs in test[0]:
        decision = 0 if obs[index] <= value else 1
        pred.append(classifier[decision])

    return pred

def classifyCART(train, test):
    (index, value) = bestSplit(train, "CART")
    counts, probs, classes = doSplit(train, index, value)
    classifier = [np.argmax(probs[0]), np.argmax(probs[1])]


    pred = []
    for obs in test[0]:
        decision = 0 if obs[index] <= value else 1
        pred.append(classifier[decision])

    return pred
```

2. **Support Vector Machines** Recall that we have learned about hard-margin linear SVM (with no slack allowed), soft-margin linear SVM (slack is allowed), and soft-margin kernel SVM in class. Suppose that we want to solve a problem of hard-margin kernel SVM. In other words, if ϕ is our mapping from the input space to the feature space (for each input point x_i we have $\phi(x_i)$ in feature space); $K(x_i, x_j)$ denote the kernel function; and w is the weight vector in feature space, we want to find $h(\phi(x)) : w^T\phi(x) = 0$. Since we

are looking for a hard-margin kernel SVM, our objective funstion would be:

$$\begin{aligned} \min\ J(w) &= \frac{1}{2}\|w\|^2 \\ \text{s.t. } \forall(x_i, y_i),\ y_i(w^T\phi(x_i)) &\geq 1 \end{aligned} \tag{4}$$

In this homework, you will find a solution to this problem following these steps:

(a) **[10 pts.]** Using Lagrange multipliers for the constraints, write the Lagrangian objective function for this problem. What new constraints are added to this objective function?
Solution:
After using Lagrange multipliers the new objective function is:

$$min\ L = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i[y_i(w^T\phi(x_i)) - 1]$$

The new constraints should be added is $\alpha_i > 0$.

(b) **[10 pts.]** Write the derivative of the the Lagrangian objective function, that you obtained in previous section with respect to w.
Solution:
The derivatieve of the Lagrangian objective function is:

$$\frac{\partial}{\partial w}L = w - \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$$

(c) **[10 pts.]** Set the derivatives of previous section to 0 and solve for w. What do you achieve? How do you interpret these results?
Solution:
Set the derivatives $\frac{\partial}{\partial w}L$ in section (b) to 0, we can obtain:

$$\frac{\partial}{\partial w}L = 0 \implies w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$$

We can see that $\mathbf{w}$ can be expressed as a linear combination of $\phi(\mathbf{x_i})$, $\phi(x_i)$ are data points x_i in feature space mapped by some nonlinear transformation ϕ, with the signed Lagrange multipliers, $\alpha_i yi$, serving as the coefficients.

(d) **[10 pts.]** Replace the w, obtained in part (c) in the objective function in part (b). Use your conclusions in part (c) to rewrite the objective function. The new objective function, dual Lagrangian, should be defined in terms of Lagrange multipliers only. What are the constraints that are added to the dual Lagrangian objective function?
Solution:
Replace the w in equation of section a with $w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$, the dual Lagrangian objective function is:

$$\begin{aligned} \max_{\alpha} \quad L_{dual} &= \frac{1}{2}(\sum_{i=1}^{n}(\alpha_i y_i \phi(x_i))^2 - \sum_{i=1}^{n} \alpha_i[y_i((\sum_{i=1}^{n} \alpha_i y_i \phi(x_i))^T \phi(x_i)) - 1] \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad s.t\, \alpha_i > 0\, and\, \alpha_j > 0 \end{aligned}$$

The constraints that are added to the dual Lagrangian objective function are $\alpha_i > 0$ and $\alpha_j > 0$
And $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

(e) **[10 pts.]** Suppose that we want to solve the dual Lagrangian objective function, to obtain the Lagrange multipliers, using gradient ascent algorithm. Remember that gradient ascent is an iterative algorithm that updates the parameters, according to the gradient of loss with respect to the parameters, in each iteration. In other words, if the $J(\alpha)$ is the objective function, in iteration $t+1$ we have $\alpha_{t+1} = \alpha_t + \text{step} * \frac{\partial J(\alpha)}{\partial \alpha}$, in which step is a constant. As a first step to do so, calculate the partial gradient of the dual Lagrangian objective function with respect to one of the Lagrange multipliers. Write the gradient in terms of the kernel function.

Solution:

The partial gradientof the dual Lagrangian objective function with respect to one of the Lagrange multiplier is:

$$J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$$

$$\implies \frac{\partial J(\alpha)}{\partial \alpha_k} = 1 - y_k (\sum_{i=1}^{n} \alpha_i y_i K(x_i, x_k))$$

(f) **[10 pts.]** Suppose that we used the result of part (e) to calculate all of the Lagrange multipliers. How can we use them to calculate w? How can we classify any new point z?

Solution:

By using gradient ascent algorithm which gradient is what we get in part e, after several iterations, $J(\alpha)$ will convergence. We then can get a best α, then use this α to classify the new point z by:

$$\hat{y} = sign(w^T \phi(z)) = sign(\sum_{\alpha_i > 0} \alpha_i y_i K(x_i, z))$$

(g) **[10 pts.]** Assume that we chose a kernel function and found hard-margin kernel SVM for some data points according to some training data. Which of the following pictures can represent a hard-margin kernel SVM? Explain why, or why not, for each.

Solution:

a) can not represent a hard-margin kernel SVM, because there are two mis-classified data point one is a orange point in blue class and also a blue data point in orange class.

b) is hard-margin kernel SVM, because two classes are perfectly separate, but the kernel is linear.

c) is hard-margin kernel SVM. because two classes are perfectly separate.

d) is not hard-margin kernel SVM, because there are two data point lie on the hyperplane.