CSEP 546 Data Mining/Machine Learning, Winter 2014: Homework 3

Due: Monday, February 17^{th}, beginning of class

1 Naïve Bayes[28 points]

The following table contains data from an employee database. The database includes the status, department, age range and salary of each employee.

Status	Department	Age	Salary
Senior	Sales	31-35	46K-50K
Junior	Sales	26-30	26K-30K
Junior	Sales	31-35	31K-35K
Junior	Systems	21-25	46K-50K
Senior	Systems	36-40	66K-70K
Junior	Systems	26-30	66K-70K
Senior	Systems	41-45	66K-70K
Senior	Marketing	36-40	46K-50K
Junior	Marketing	31-35	41K-45K
Senior	Secretary	46-50	36K-40K
Junior	Secretary	26-30	26K-30K

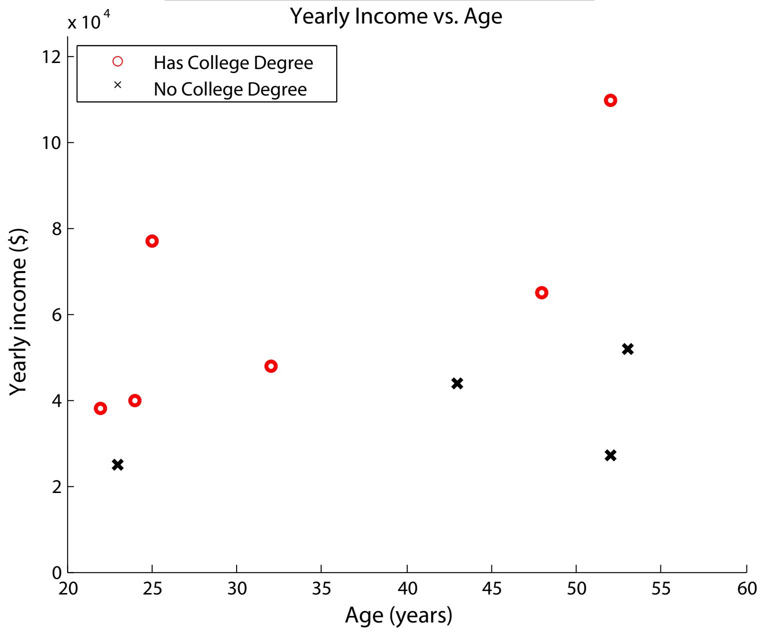
This problem asks you to learn a Naïve Bayes classifier for predicting the employee status.

1. *(4 points)* What is the prior probability of status $p(status)$?
2. *(10 points)* What is the conditional probability (given status) for department, age and salary respectively? Please write down your answers in three tables for the three conditional probabilities, respectively.
3. *(4 points)* Use your naïve Bayes classifier, predict the status for two instances A={Marketing, 31-35, 46K-50K} and B={Sales, 31-35, 66K-70K}.
4. *(6 points)* Suppose we add another feature called "SalaryDuplicate", which takes on the same value as "Salary" for all training examples. What are the prediction results for the above two instances, if we train a naïve Bayes classifier on the same dataset with this extra feature?
5. *(6 points)* Why do you observe the differences in Questions 1.3 and 1.4? What property of the naïve Bayes classifier was affected?

2 Decision Trees [22 points]

1. *(10 points)* Consider the problem of predicting if a person has a college degree based on age and salary. The table and graph below contain training data for 10 individuals.

Age	Salary ($)	College Degree
24	40,000	Yes
53	52,000	No
23	25,000	No
25	77,000	Yes
32	48,000	Yes
52	110,000	Yes
22	38,000	Yes
43	44,000	No
52	27,000	No
48	65,000	Yes



Build a decision tree for classifying whether a person has a college degree by greedily choosing threshold splits that maximize information gain. What is the depth of your tree and the information gain at each split?

2. (6 points) A multivariate decision tree is a generalization of univariate decision trees, where more than one attribute can be used in the decision rule for each split. That is, splits need not be orthogonal to a feature's axis.

   For the same data, learn a multivariate decision tree where each decision rule is a linear classifier that makes decisions based on the sign of $\alpha x_{age} + \beta x_{income} - 1$.

   What is the depth of your tree, as well as α, β and the information gain for each split?

3. (6 points) Multivariate decision trees have practical advantages and disadvantages. List two advantages and two disadvantages multivariate decision trees have compared to univariate decision trees.

3 Programming Question [50 points]

3.1 Implement Adaboost on decision stumps.

Given the data $\mathcal{D} = \{(x_i, y_i)\}_{i=0}^{N-1},\ x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$, the Adaboost algorithm uses an ensemble of weak classifiers to build a more expressive one. Recall that the algorithm itself is given by:

```
$\mathcal{D} \leftarrow \{(\mathbf{x}_i, y_i)\}_{i=0}^{N-1}$
$D_0[0:N] \leftarrow 1/N$
**for** $t = 0 \rightarrow T-1$ **do**
    $h_t \leftarrow \arg\max_{h \in \mathcal{H}} |0.5 - \epsilon(h, D_t)|$
    $\alpha_t \leftarrow \frac{1}{2}\ln(\frac{1-\epsilon(h_t,D_t)}{\epsilon(h_t,D_t)})$
    $D_{t+1}[0:N] = \frac{1}{Z} D_t[0:N] * \mathrm{e}^{\alpha_t(2\,\mathbb{1}(y[0:N] \neq h_t(\mathbf{x}[0:N]))-1)}$
**end for**
**return** $\{\alpha_t, h_t\}$.
```

In the above algorithm, Z is the normalizer of D_{t+1}, T is the maximum number of weak-classifiers in the ensemble, and

$$\epsilon(h, D) = \sum_{i=0}^{N-1} D[i]\,\mathbb{1}(y_i \neq h(\mathbf{x}_i)).$$

The final classifier is given by,

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=0}^{T} \alpha_t h_t(\mathbf{x})\right).$$

For this assignment you'll be implementing Adaboost to run on decision-stumps. A decision stump is a binary classifier given by,

$$h(\mathbf{x}, i, v) = 2\,\mathbb{1}(\mathbf{x}[i] > v) - 1.$$

Decision stumps can be trained in $O(\kappa N)$ [1] time by maximizing mutual information. Note that the family of weak classifiers in this case is finite; there are $N-1$ classifiers for each feature, hence effectively $|\mathcal{H}| = (N-1)d$.

You are provided with implementations for training decision stumps on both sparse and dense data. A summary of the usage of these routines is given below.

[1] The κ here refers to the sparsity of the data.

```
#!/usr/bin/python2
from scipy import *
import scipy.sparse as sp
import dstump as ds

(f, y) = ds.two_clusters(100)
pr = ones(len(y))/len(y)

#This quantity is invariant for each Adaboost step, and helps us take
#advantage of sparsity.
pplus = sum(pr * (y > 0))

#The decision stump training routine accepts either a dense 1-d
#array or a sparse 1-d CSC matrix. The resulting decision variable
#might be different for dense and sparse data, but the errors are
#the same. See implementation for details.
(dv, err) = ds.stump_fit(f, y, pr, pplus)

#Inplace transpose of a CSR matrix gives a CSC matrix.
fs = sp.csr_matrix(f).T
(dvs, errs) = ds.stump_fit(fs, y, pr, pplus)
```

3.2 Try breaking your implementation.

You'll now test your implementation on synthetically generated datasets [2]. For each of the functions, {`two_lines`, `four_clusters`} defined in `adaboost-test.py`:

1. (*5 × 2 points*) Generate the training dataset, with $N := 500$ data points using the function. Run Adaboost on this dataset for about $T := 50$ steps [3]. Plot the training error $\epsilon_t = \frac{1}{N}\sum_{i=0}^{N-1}(y_i \neq H_t(\mathbf{x}_i))$, for every step of Adaboost. Also plot the magnitude of the coefficient α_t, for each step, in a separate graph.
2. (*4 × 2 points*) Generate the validation dataset, with $N_v := 500$ data points using the same function. You'll now use this dataset to pick the size of the ensemble. We'll denote by H_t, the classifier obtained at the end of the t'th step of Adaboost.

$$H_t(\mathbf{x}) = \text{sign}\left(\sum_{j=0}^{t}\alpha_j h_j(\mathbf{x})\right).$$

   Using the model parameters learnt from (1), plot the prediction error ϵ_t for every classifier $H_t,\ t = \{0 \dots m\}$.

$$\epsilon_t = \frac{1}{N_v}\sum_{i=0}^{N_v-1}(y_i \neq H_t(\mathbf{x}_i)).$$

3. (*1 × 2 points*) Generate the test dataset, with $N_t := 500$ data points using the same function. For the value of t^* that gives the minimum validation error in (2), compute the prediction error on the test data. Is this better than random ? (*Extra credit: 5 points*) If not, can you explain why ?

Please answer the above set of questions, separately for each test function.

3.3 Let's classify newsgroups.

We'll now use Adaboost to classify emails between two newsgroups. For training the classifier we'll be making use of the 20-Newsgroups dataset, which consists of an archive of about 1000 posts from 20 newsgroups each.

To convert text into numerical data, we'll be using a bag-of-words tokenizer. A bag-of-words tokenizer keeps count of the occurence of particular words or sequence of words, irrespective of their position in a document. Naive as this may be, the features hence extracted can be surprisingly effective at classifiying documents. Since some words occur frequently irrespective of the subject - words like "the" - we'll also weight the frequencies by how infrequent they are in the text corpus. This scheme is known popularly as "tf-idf".

[2] You can use matplotlib.pyplot.plot to plot the features in 2-D.
[3] A "step" here would be one run of the Adaboost loop.

We'll be making use of scikits-learn for tokenizing the text. You're provided with a text parser (`tok.py`), which strips out comments and message-headers from emails. You're also provided with some starter code (`scikits-starter.py`) to help get you familiarized with `sklearn.feature_extraction.text`.

You are now tasked with parsing the emails from a given pair of newsgroups, and then using Adaboost to learn a binary classifier from them. For each of the following pairs of newsgroups, {["alt.atheism", "comp.windows.x"], ["alt.atheism", "soc.religion.christian"]},

- Parse the text of emails from each newsgroup in the pair to create the corpus. Split the text corpus in proportions (0.6, 0.2, 0.2) for training, validation and testing respectively.
- Use the tf-idf tokenizer in scikits-learn, with the parameters given in the starter code, to fit a bag-of-words model on the training corpus.
- Run Adaboost for about $T := 30$ steps. This may take a considerable amount time ($\sim$ 10m).
- Answer the following questions, which are in the spirit of the previous section.

1. (*7 × 2 points*) Plot the training error for each step of Adaboost. Also plot the magnitude of the coefficient α_t, for every step, in a separate graph.
2. (*4 × 2 points*) Find the optimal ensemble size using the validation dataset. Report the test error for the resulting classifier. Show a plot of the validation error *vs* the number of weak-learners used.
3. (*2 × 2 points*) Report the feature names for the classifier obtained in (2).
4. (*2 × 2 points*) For each newsgroup in the 20-newsgroups dataset, which one amongst the given pair, is your classifier from (2) likely to classify emails from the newsgroup, into (use majority-vote) [4]? For example, if your classifier was trained on ["alt.atheism", "comp.windows.x"], and you found that 90% of the emails from "soc.religion.christian" were classified as belonging to "alt.atheism", then you'd report "alt.atheism (0.9)" against "soc.religion.christian" in a table.

As before, answer the above set of questions, separately for each pair.

[4] Any Yes Minister fans out there?