CIS 419/519 Introduction to Machine Learning
Assignment 3

Due: November 1, 2017 11:59pm

Instructions

Read all instructions in this section thoroughly.

Collaboration: Make certain that you understand the course collaboration policy, described on the course website. You must complete this assignment **individually**; you are **not** allowed to collaborate with anyone else. You may *discuss* the homework to understand the problems and the mathematics behind the various learning algorithms, but **you are not allowed to share problem solutions or your code with any other students.** You must also not consult code on the internet that is directly related to the programming exercise. We will be using automatic checking software to detect academic dishonesty, so please don't do it.

You are also prohibited from posting any part of your solution to the internet, even after the course is complete. Similarly, please don't post this PDF file or the homework skeleton code to the internet.

Formatting: This assignment consists of two parts: a problem set and program exercises.

For the problem set, you must write up your solutions electronically and submit it as a single PDF document. We will not accept handwritten or paper copies of the homework. Your problem set solutions must use proper mathematical formatting. For this reason, we **strongly** encourage you to write up your responses using LaTeX. (Alternative word processors, such as MS Word, produce very poorly formatted mathematics.)

Your solutions to the programming exercises must be implemented in python, following the precise instructions included in Part 2. Portions of the programing exercise will be graded automatically, so it is imperative that your code follows the specified API. A few parts of the programming exercise asks you to create plots or describe results; these should be included in the same PDF document that you create for the problem set.

Homework Template and Files to Get You Started: The homework zip file contains the skeleton code and data sets that you will require for this assignment. **Please read through the documentation provided in ALL files before starting the assignment.**

Citing Your Sources: Any sources of help that you consult while completing this assignment (other students, textbooks, websites, etc.) ***MUST*** be noted in the your `README` file. This includes anyone you briefly discussed the homework with. If you received help from the following sources, you do not need to cite it: course instructor, course teaching assistants, course lecture notes, course textbooks or other readings.

Submitting Your Solution: We will post instructions for submitting your solution one week before the assignment is due. Be sure to check Piazza then for details.

CIS 519 ONLY Problems: Several problems are marked as "[CIS 519 ONLY]" in this assignment. Only students enrolled in CIS 519 are required to complete these problems. However, we do encourage students in CIS 419 to read through these problems, although you are not required to complete them.

All homeworks will receive a percentage grade, but CIS 519 homeworks will be graded out of a different total number of points than CIS 419 homeworks. Students in CIS 419 choosing to complete CIS 519 ONLY exercises will not receive any credit for answers to these questions (i.e., they will not count as extra credit nor will they compensate for points lost on other problems).

Assignment Version 20171019

PART I: PROBLEM SET

Your solutions to the problems will be submitted as a single PDF document. Be certain that your problems are well-numbered and that it is clear what your answers are. Additionally, you will be required to duplicate your answers to particular problems in the `README` file that you will submit.

1 Probability decision boundary (10pts)

Consider a case where we have learned a conditional probability distribution $P(y \mid \mathbf{x})$. Suppose there are only two classes, and let $p_0 = P(y = 0 \mid \mathbf{x})$ and let $p_1 = P(y = 1 \mid \mathbf{x})$. A loss matrix gives the cost that is incurred for each element of the confusion matrix. (E.g., true positives might cost nothing, but a false positive might cost us \$10.) Consider the following loss matrix:

	$y = 0$ (true)	$y = 1$ (true)
$\hat{y} = 0$ (predicted)	0	10
$\hat{y} = 1$ (predicted)	5	0

(a) Show that the decision $\hat{y}$ that minimizes the expected loss is equivalent to setting a probability threshold θ and predicting $\hat{y} = 0$ if $p_1 < \theta$ and $\hat{y} = 1$ if $p_1 \geq \theta$.

(b) What is the threshold for this loss matrix?

2 Double counting the evidence (15pts)

Consider a problem in which the binary class label $Y \in \{T, F\}$ and each training example $\mathbf{x}$ has 2 binary attributes $X_1, X_2 \in \{T, F\}$.

Let the class prior be $p(Y = T) = 0.5$ and $p(X_1 = T \mid Y = T) = 0.8$ and $p(X_2 = T \mid Y = T) = 0.5$. Likewise, $p(X_1 = F \mid Y = F) = 0.7$ and $p(X_2 = F \mid Y = F) = 0.9$. Attribute X_1 provides slightly stronger evidence about the class label than X_2.

Assume X_1 and X_2 are truly independent given Y. Write down the naive Bayes decision rule.

(a) What is the expected error rate of naive Bayes if it uses only attribute X_1? What if it uses only X_2?

The expected error rate is the probability that each class generates an observation where the decision rule is incorrect. If Y is the true class label, let $\hat{Y}(X_1, X_2)$ be the predicted class label. Then the expected error rate is $p(X_1, X_2, Y \mid Y \neq \hat{Y}(X_1, X_2))$.

(b) Show that if naive Bayes uses both attributes, X_1 and X_2, the error rate is 0.235, which is better than if using only a single attribute (X_1 or X_2).

(c) Now suppose that we create new attribute X_3 that is an exact copy of X_2. So for every training example, attributes X_2 and X_3 have the same value. What is the expected error of naive Bayes now?

(d) Briefly explain what is happening with naive Bayes (2 sentences max).

(e) Does logistic regression suffer from the same problem? Briefly explain why (2 sentences max).

3 Reject option (CIS 519 ONLY – 10pts)

In many applications, the classifier is allowed to "reject" a test example rather than classifying it into one of the classes. Consider, for example, a case in which the cost of misclassification is \$10 but the cost of having a human manually make the decision is only \$3. We can formulate this as the following loss matrix:

	$y = 0$ (true)	$y = 1$ (true)
$\hat{y} = 0$ (predicted)	0	10
$\hat{y} = 1$ (predicted)	10	0
reject	3	3

(a) Suppose $p(y = 1|\mathbf{x}) = 0.2$. Which decision minimizes the expected loss?

(b) Now suppose $p(y = 1|\mathbf{x}) = 0.4$. Now which decision minimizes the expected loss?

(c) Show that in cases such as this there will be two thresholds, θ_0 and θ_1, such that the optimal decision is to predict 0 if $p_1 < \theta_0$, reject if $\theta_0 \leq p_1 \leq \theta_1$, and predict 1 if $p_1 > \theta_1$.

(d) What are the values of these thresholds for the following loss matrix?

	$y = 0$ (true)	$y = 1$ (true)
$\hat{y} = 0$ (predicted)	0	10
$\hat{y} = 1$ (predicted)	5	0
reject	3	3

PART II: PROGRAMMING EXERCISES

1 Challenge: Generalizing to Unseen Data (30 pts)

One of the most difficult aspects of machine learning is that your classifier must generalize well to unseen data. In this exercise, we are supplying you with labeled training data and *unlabeled* test data. Specifically, you will *not* have access to the labels for the test data, which we will use to grade your assignment. You will fit the best model that you can to the given data and then use that model to predict labels for the test data. It is these predicted labels that you will submit, and we will grade your submission based on your test accuracy (relative to the best performance you should be able to obtain). Each instance belongs to one of nine classes, named '1' . . . '9'. We will not provide any further information on the data set.

You will submit two sets of predictions: one based on a boosted decision tree classifier (which you will write), and another set of predictions based on whatever machine learning method you like – you are free to choose any classification method. We will compute your test accuracy based on your predicted labels for the test data and the true test labels. Note also that we will not be providing any feedback on your predictions or your test accuracy when you submit your assignment, so you must do your best without feedback on your test performance.

Relevant Files in the Homework Skeleton[1]

- **boostedDT.py**
- `test_boostedDT.py`
- `data/challengeTrainLabeled.dat`: labeled training data for the challenge
- `data/challengeTestUnlabeled.dat`: unlabeled testing data for the challenge

1.1 The Boosted Decision Tree Classifier

In class, we mentioned that boosted decision trees have been shown to be one of the best "out-of-the-box" classifiers. (That is, if you know nothing about the data set and can't do parameter tuning, they will likely work quite well.) Boosting allows the decision trees to represent a much more complex decision surface than a single decision tree.

Write a class that implements a boosted decision tree classifier. Your implementation may rely on the decision tree classifier already provided in scikit_learn (`sklearn.tree.DecisionTreeClassifier`), but you must implement the boosting process yourself. (The scikit_learn module actually provides boosting as a meta-classifier, but you may not use it in your implementation.) Each decision tree in the ensemble should be limited to a maximum depth as specified in the `BoostedDT` constructor. You can configure the maximum depth of the tree via the `max_depth` argument to the `DecisionTreeClassifier` constructor.

[1] **Bold text** indicates files that you will need to complete; you should not need to modify any of the other files.

Your class must implement the following API:

- `__init__(numBoostingIters = 100, maxTreeDepth = 3)`: the constructor, which takes in the number of boosting iterations (default value: 100) and the maximum depth of the member decision trees (default: 3)
- `fit(X,y)`: train the classifier from labeled data (X, y)
- `predict(X)`: return an array of n predictions for each of n rows of X

Note that these methods have already been defined correctly for you in `boostedDT.py`; be very careful not to change the API. You should configure your boosted decision tree classifier to be the best "out-of-the-box" classifier you can; you may not modify the constructor to take in additional parameters (e.g., to configure the individual decision trees).

There is one additional change you need to make to AdaBoost beyond the algorithm described in class. AdaBoost by default only works with binary classes, but in this case, we have a multi-class classification problem. One variant of AdaBoost, called AdaBoost-SAMME, easily adapts AdaBoost to multiple classes. Instead of using the equation $\beta_t = \frac{1}{2}\ln\left(\frac{1-\epsilon}{\epsilon}\right)$ in AdaBoost, you should use the AdaBoost-SAMME equation

$$\beta_t = \frac{1}{2}\left(\ln\left(\frac{1-\epsilon}{\epsilon}\right) + \ln(K-1)\right) \quad ,$$

where K is the total number of classes. This will force $\beta_t \geq 0$ as long as the classifier is worse than random guessing (in this case random guessing would be $1/K$, so the error rate would need to be greater than $1 - 1/K$). Note that when $K = 2$, AdaBoost-SAMME reduces to AdaBoost. For further information on SAMME, see http://web.stanford.edu/~hastie/Papers/samme.pdf.

Test your implementation by running `test_boostedDT.py`, which compares your `BoostedDT` model to a regular decision tree on the iris data with a 50:50 training/testing split. You should see that your `BoostedDT` model is able to obtain ∼97.3% accuracy vs the 96% accuracy of regular decision trees. Make certain that your implementation works correctly before moving on to the next part.

Once your boosted decision tree is working, train your `BoostedDT` on the labeled data available in the file `data/challengeTrainLabeled.dat`. The class labels are specified in the last column of data. You may tune the number of boosting iterations and maximum tree depth however you like. Then, use the trained `BoostedDT` classifier to predict a label $y \in \{1, \ldots, 9\}$ for each unlabeled instance in `data/challengeTestUnlabeled.dat`. Your implementation should output a comma-separated list of predicted labels, such as

```
1, 2, 1, 9, 4, 1, 3, 1, 5, 3, 4, 2, 8, 3, 1, 6, 3, ...
```

Be very careful not to shuffle the instances in `data/challengeTestUnlabeled.dat`; the first predicted label should correspond to the first unlabeled instance in the testing data. The number of predictions should match the number of unlabeled test instances.

Copy and paste this comma-separated list into the README file to submit your predictions for grading. Also, record the expected accuracy of your model in the README file. Finally, also save the comma-separated list into a text file named `predictions-BoostedDT.dat`; this file should have exactly one line of text that contains the list of predictions.

1.2 Training the Best Classifier

Now, train the very best classifier for the challenge data, and use that classifier to output a second vector of predictions for the test instances. You may use any machine learning algorithm you like, and may tune it any way you wish. You may use the method and helper functions built into scikit_learn; you do not need to implement the method yourself, but may if you wish. If you don't want to use scikit_learn, you may use any other machine learning software you wish. If you can think of a way that the unlabeled data in `data/challengeTestUnlabeled.dat` would be useful during the training process, you are welcome to let your classifier have access to it during training.

Note that you will not be submitting an implementation of your optimal model, just its predictions.

Once again, use your trained model to output a comma-separated list of predicted labels for the unlabeled instances in `data/challengeTestUnlabeled.dat`. Again, be careful not to shuffle the test instances; the order of the predictions must match the order of the test instances.

Copy and paste this comma-separated list into the README file to submit your predictions for grading. Also, record the expected accuracy of your model in the README file. Finally, also save the comma-separated list into a text file named `predictions-BestClassifier.dat`; this file should have exactly one line of text that contains the list of predictions.

If you believe that your boostedDT classifier (from the previous section) is actually the best set of predictions for this challenge data, then you would submit the boostedDT predictions twice in the README file (and have two identical files of predictions).

Write a brief paragraph (3–4 sentences max) describing the best machine learning classifier you found, its optimal parameter settings (if any), and how you trained the model. Include that paragraph in your PDF writeup, and also in the README.

2 What's Cooking, Naïve Bayes? (20 pts for 419, 30 pts for 519)

Join the What's Cooking competition on Kaggle: https://www.kaggle.com/c/whats-cooking. Download the training and test data (in .json). The competition page describes how these files are formatted.

Tell us about the data in a well-formatted table that includes: the number of dishes in the training set, the number and types of cuisine, the number of unique ingredients, etc.

Represent each dish by a binary ingredient feature vector. Suppose there are d different ingredients in total. Represent each dish by a d-dimensional binary ingredient vector $\boldsymbol{x}$, where $x_i = 1$ if the dish contains ingredient i and $x_i = 0$ otherwise. For example, suppose all the ingredients we have in the training set are $\{\text{beef}, \text{chicken}, \text{egg}, \text{lettuce}, \text{tomato}, \text{rice}\}$ and the dish is made by ingredients $\{\text{chicken}, \text{lettuce}, \text{tomato}, \text{rice}\}$, then the dish could be represented by a 6-dimensional binary vector $[0, 1, 0, 1, 1, 1]$ as its features. Create matrices for both the training and testing data, and store these data sets in two files called `cooking_train.dat` and `cooking_test.dat`. You are welcome to process this data and create the data files however you like; you do not need to write a python script to do the processing automatically, but it might be helpful.

Write a program called `whatsCooking.py` that 1) reads in the training data set, 2) uses sklearn's implementations of the Naïve Bayes and Logistic Regression classifiers to perform 10-trials of 10-fold cross-validation on the training data, and 3) outputs the mean and standard error of the accuracy of each classifier in a neat table. For Naïve Bayes, try both the Gaussian distribution prior assumption (naive_bayes.GaussianNB) and Bernoulli distribution prior assumption (naive_bayes.BernoulliNB), so your script will be comparing three classifiers in total. Be sure to tune any free parameters of each classifier. You are welcome to use sklearn's built-in routines for doing tuning and cross-validation; you do not need to implement these yourself.

Include the output table in your PDF writeup. For the Gaussian prior and Bernoulli prior, which performs better in terms of cross-validation accuracy? Why? Please give specific arguments. Discuss how Logistic Regression performs in comparison.

Train your best-performing classifier with all of the training data, and generate test labels on the test set. Submit your results to Kaggle and report the accuracy you obtained in your PDF writeup. Also describe your best-performing classifier, including the values you used for any tuned parameter.

Training Your Own Classifier (519 Only)

Duplicate your script `whatsCooking.py` into a new file called `whatsCooking_MyClassifier.py` and modify it to train your own model instead of Logistic Regression and Naïve Bayes. You are welcome to use any implementation of any classifier provided by sklearn, and tune it however you like. Train the best model that you can for this competition, tuning all free parameters, and then generate test labels on the test set. Submit your results to Kaggle and report the accuracy you obtained in your PDF writeup. Describe your own classifier, including the values you used for any tuned parameters.