CSE 446: Machine Learning
Assignment 3

Due: March 4th, 2020 9:30am

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.

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 programming exercises.

For the problem set, you must write up your solutions electronically and submit it as a single PDF document. Your problem set solutions must use proper mathematical formatting. For this reason, we **strongly** encourage you to write up your responses using LaTeX.

Your solutions to the programming exercises must be implemented in python, following the precise instructions included in Part 2. Several parts of the programming exercise ask 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 PDF document. 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 course materials.

Submitting Your Solution: You will be submitting only the following files, which you created or modified as part of homework 3:

- `hw3-UWNETID.pdf` (a PDF of your homework 3 writeup)
- `boostedDT.py`
- `bestClassifier.py`
- `predictions-BestClassifier.dat`
- `predictions-BoostedDT.dat`
- `naiveBayes.py`

Please follow the naming conventions exactly, and do not submit additional files including the test scripts or data sets. Your PDF writeup of Homework w should be named `hww-UWNETID.pdf`, where "UWNETID" is your own UW netID (for example, my file would be named "hw3-bboots.pdf"). Please submit both the PDF and the `.py` files through Gradescope.

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.

1 Kernels (10pts)

Suppose that our inputs are one dimensional and that our feature map is infinite dimensional: $\phi(x)$ is a vector whose ith component is

$$\frac{1}{\sqrt{i!}} e^{-\frac{x^2}{2}} x^i.$$

for all nonnegative integers i. (Thus, ϕ is an infinite dimensional vector.) Show that $K(x, x') = e^{-\frac{(x-x')^2}{2}}$ is a kernel function for this feature map, i.e.,

$$\phi(x) \cdot \phi(x') = e^{-\frac{(x-x')^2}{2}}.$$

Hint: Use the Taylor expansion of e^z. (This is the one dimensional version of the Gaussian (RBF) kernel).

2 Probability decision boundary (5pts)

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 $p_1 = P(y = 1 \mid \mathbf{x})$, and additionally assume that $P(p_1 < q) = q$ for any value $q \in [0, 1]$. 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) First we prove 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 $p_1 \geq \theta$.

Proof:

A decision rule is a mapping from $p_1 \in [0, 1]$ to $\hat{y} \in \{0, 1\}$. First we observe that, our decision being threshold as above, namely,

$$\hat{y} = 0 \text{ if } p_1 < \theta \text{ and } \hat{y} = 1 \text{ if } p_1 \geq \theta$$

is equivalent with (if and only if), for any pairs of $\theta_1, \theta_2 \in [0, 1]$ such that $\theta_1 < \theta_2$, it is *never* the case that

$$\hat{y} = 1 \text{ if } p_1 = \theta_1 \text{ and } \hat{y} = 0 \text{ if } p_1 = \theta_2$$

Now we are just left to show that for any arbitrary pair θ_1, θ_2, there always exists a decision rule that has less expected loss than

rule 1: $\hat{y} = 1$ if $p_1 = \theta_1$; $\hat{y} = 0$ if $p_1 = \theta_2$; $\hat{y} = f(p_1)$ if $p_1 \neq \theta_1$ and $p_1 \neq \theta_2$

where my f is any mapping from $p_1 \in [0, 1]$ to $\hat{y} \in \{0, 1\}$ except on the two points.

First we state two alternative decision rules:

rule 2: $\hat{y} = 0$ if $p_1 = \theta_1$; $\hat{y} = 0$ if $p_1 = \theta_2$; $\hat{y} = f(p_1)$ if $p_1 \neq \theta_1$ and $p_1 \neq \theta_2$

rule 3: $\hat{y} = 1$ if $p_1 = \theta_1$; $\hat{y} = 1$ if $p_1 = \theta_2$; $\hat{y} = f(p_1)$ if $p_1 \neq \theta_1$ and $p_1 \neq \theta_2$

where both of the rules above shares f from rule 1, meaning they will do exactly the same with rule 1 except on the two points, θ_1 and θ_2.

Now, if we consider the costs on only the two points, we get the following:
rule 1: $5(1 - \theta_1)$ when $p_1 = \theta_1$ and $10\theta_2$ when $p_1 = \theta_2$
rule 2: $10\theta_1$ when $p_1 = \theta_1$ and $10\theta_2$ when $p_1 = \theta_2$
rule 3: $5(1 - \theta_1)$ when $p_1 = \theta_1$ and $5(1 - \theta_2)$ when $p_1 = \theta_2$

We now claim that at least one of rule 2 and rule 3 has a lower cost than rule 1, which is equivalent from saying, if $\theta_1 < \theta_2$, then either $5(1-\theta_1) > 10\theta_1$ or $10\theta_2 > 5(1-\theta_2)$, or both.

We prove this by contradiction. The negation of the above statement says, if $\theta_1 < \theta_2$, then $5(1-\theta_1) \leq 10\theta_1$ and $10\theta_2 \leq 5(1-\theta_2)$. However, $5(1-\theta_1) \leq 10\theta_1 \iff \theta_1 \geq \frac{1}{3}$ and $10\theta_2 \leq 5(1-\theta_2) \iff \theta_2 \leq \frac{1}{3}$, which directly contradicts our assumption that $\theta_1 < \theta_2$. Therefore, the statement from above must be true. This completes the proof.

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

3 Double counting the evidence (20pts)

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 $\mathbb{E}_{X_1,X_2,Y}[\mathbf{1}\{Y \neq \hat{Y}(X_1, X_2)\}]$, where $\mathbf{1}\{\cdot\}$ is the indicator function that takes in an event, returns 1 if it's true and 0 otherwise.

(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).

4 Reject option (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) Observe 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 $p_1 > \theta_1$. **This problem no longer requires you to do anything, you may just assume the statement is true now. The proof of the statement should be similar to the proof in 2(a).**

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

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.

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 no worse than random guessing. 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% 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.

Record the expected accuracy of your model in the PDF 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.

Please submit your implementation in a file called `bestClassifier.py`, along with the 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.

Record the expected accuracy of your model in your PDF file. Also save the comma-separated list into a text file named `predictions-BestClassifier.dat`; this file should have one line of text that contains the list of predictions.

Write a brief paragraph (6–8 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.

2 Naive Bayes (35 pts)

In this implementation exercise, you will implement naive Bayes for batch learning.

Relevant Files in the Homework Skeleton[2]

- **naiveBayes.py**
- `test_naiveBayes.py`
- `test_onlineNaiveBayes.py`

2.1 Implementing Batch Naive Bayes

Implement a **multinomial** naive Bayes classifier in the `NaiveBayes` class in `naiveBayes.py`. Your implementation should support Laplace smoothing. Whether or not to use Laplace smoothing is controlled via an argument to the constructor; Laplace smoothing is enabled by default. Your implementation must follow the API below. (Note that all matrices are actually 2D numpy arrays in the implementation.)

- `__init(useLaplaceSmoothing=True)__` : constructor
- `fit(X,Y)`: method to train the naive Bayes model
- `predict(X)`: method to use the trained naive Bayes model for prediction
- `predictProbs(X)`: outputs a matrix of predicted posterior class probabilities

The training data for multinomial naive Bayes is specified as feature counts: `X[i,j]` is the number of times feature j occurs in instance i (or you can think of it as that instance i is characterized by a particular real-valued amount of feature j).

We are here using the multinomial distribution. Suppose our dataset has d features and K classes. Then the multinomial distribution for a particular sample $(\mathbf{x}, y)$, where $\mathbf{x}_i \in \mathbb{N}$ for $i = 1, \ldots, d$ and $y \in \{1, 2, \ldots, K\}$, is

$$P(\mathbf{x}, y) = P(\mathbf{x}|y)P(y) = \underbrace{\frac{(\sum_k x_k)!}{x_1!x_2!\cdots x_d!} p_{y1}^{x_1} p_{y2}^{x_2} \cdots p_{yd}^{x_d}}_{P(\mathbf{x}|y)} \cdot P(y).$$

The naive Bayes assumption (that each instance is independent given the class label) is used in the sense that that we are assuming the generative process is (1) first picking a class y according to the label distribution $P(Y)$ (in this case, a categorical distribution) and then (2) generate a sequence of features, **independently**,

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

according to a multinomial distribution conditioned on the class y: $(p_{y1}, p_{y2}, \ldots, p_{yd})$ where $p_{y1} + p_{y2} + \cdots + p_{yd} = 1$. What you are given is the count of each feature generated by this process stored in a vector $(x_1, x_2, \ldots, x_d)$. This is a useful model for predicting, say, document classes, where d is size of vocabulary and c is number of document classes.

The MLE parameter estimation for this naive Bayes probabilistic model is then

$$\hat{P}(y) = \frac{\#\text{samples with label } y}{\#\text{total samples}}$$

and

$$\hat{p}_{yi} = \frac{\text{total occurrences of feature } i \text{ in samples with label } y}{\text{total occurrences of all features with label } y}.$$

When using Laplace smoothing, we estimate p_{yi} with

$$\hat{p}_{yi} = \frac{(\text{total occurrences of feature } i \text{ in samples with label } y) + 1}{(\text{total occurrences of all features with label } y) + d}.$$

During prediction, given feature count vector $\mathbf{x}$, we estimate label y's conditional probability with

$$\hat{P}(y|\mathbf{x}) \propto \hat{P}(y)\hat{p}_{y1}^{x_1}\hat{p}_{y2}^{x_2}\cdots\hat{p}_{yd}^{x_d}$$

up to normalization. You might want to implement the equation above with summation of log probabilities for better numerical stability. If you choose to do so, you would need another numerical trick below. After obtaining the log probabilities for each classes: $z = (\log \hat{P}(y = 1|\mathbf{x}), \log \hat{P}(y = 2|\mathbf{x}), \ldots, \log \hat{P}(y = K|\mathbf{x}))$, the actual probability distribution to be output is

$$\begin{aligned}
\hat{P}(y = i|\mathbf{x}) &= \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_K}} \\
&= \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_K}} \cdot \frac{e^{-\max_i z_i}}{e^{-\max_i z_i}} \\
&= \frac{e^{z_i - \max_i z_i}}{e^{z_1 - \max_i z_i} + e^{z_2 - \max_i z_i} + \cdots + e^{z_K - \max_i z_i}}.
\end{aligned}$$

You also need to subtract $\max_i z_i$ from the log probabilities for better numerical stability. (Note: the function above is called the softmax function and subtracting the max from the log probabilities before exponentiating is a common numerical trick.)

The `predictProbs(X)` function takes in a matrix X of n instances and outputs an $n \times K$ matrix of posterior probabilities. Each row i of the returned matrix represents the posterior probability distribution over the K classes for the i^{th} training instance. (Note that each row of the returned matrix will sum to 1.)

Run `test_naiveBayes.py` to test your implementation. Your naive Bayes should achieve ~89% accuracy.

2.2 Online Learning with Naive Bayes

Once your naive Bayes model is working correctly, extend it to learn *online* by completing the `OnlineNaiveBayes` class in `naiveBayes.py`. So far, we have only examined batch learning methods, which process the entire training set in one batch to produce the model. In contrast, online learning methods receive data instances consecutively, updating the model each time. In most cases, after having seen the same set of training instances, the online learning algorithm will produce the same model as the batch learning algorithm, but it will have only seen the instances one at a time.

Essentially, the idea is that the `.fit()` method will be called multiple times, once for each training instance. At any time, we can call the `.predict()` method to use the classifier to predict labels for new instances. For example,

```
...
model = OnlineNaiveBayes()
model.fit(Xtrain[1,:], ytrain[1])
model.fit(Xtrain[2,:], ytrain[2])
print model.predict(Xtest)
model.fit(Xtrain[3,:], ytrain[3])
print model.predict(Xtest)
...
```

Unlike in the batch learning setting, calling `.fit()` does not overwrite the original model, but instead it updates it with the new training instance.

For convenience, we will actually write a hybrid approach, in which one or more training instances can be fed into `.fit()`. This form will support different types of training paradigms: (a) batch learning, where `.fit()` is called once with all training instances, (b) online learning, where `.fit()` is called numerous times, once for each training instance, and (c) a hybrid approach where `.fit()` is called several times, each time with a few training instances.

Naive Bayes is particularly easy to convert for online learning, since all we're doing is updating counts in each CPT. The only tricks are figuring out how to handle Laplace smoothing as you go and how to add instances with never-before-seen labels online! (You must figure those out yourself).

Your implementation must follow the same API as `NaiveBayes`. Although you are permitted to implement it entirely separately (e.g., duplicate `NaiveBayes` and edit it to learn online), it might be a better idea to implement it as a subclass of your Naive Bayes classifier. That way, if you discover any issues in your naive Bayes classifier, you only need to edit `NaiveBayes` to fix it in both the batch and online versions of the classifier. Also, note that certain aspects, like the `.predict()` method, do not necessarily need to change between the two versions.

Run `test_onlineNaiveBayes.py` to test your implementation and compare it against your standard naive Bayes classifier. You should see that the two algorithms are able to achieve the same performance.