CS37300 Homework — Homework 2— Last update: Feb. 11, 2024
Due: Theory: Feb. 21, 2024 (6pm); Programming: Feb. 26, 2024 (6pm)

Name: Derrick Khou PUID: 0037028290

Instructions and Policy: You are allowed to study with others and use online resources for reference, however, the work you turn in must be your own. This means do not copy/paste from Stack Exchange (or from another student.) If you have worked closely with other students, provide their name(s) and a brief (at most one paragraph) description of the interaction; if we feel this oversteps the bounds, we will discuss it with you. Each student should write up their own solutions independently.

The requirements below are supposed to be followed in this and further homework assignments.

- For your theoretical submission:
  - YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK.
  - The answers MUST be submitted via Gradescope.
  - Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.
  - Theoretical questions MUST include the intermediate steps to the final answer.
- For your programming answers:
  - The Python scripts will be submitted separately via Gradescope
  - Zero points will be given in any question where the Python code answer doesn't match the answer on Gradescope.
  - If the answer is/includes a plot, it should be added to your theoretical submission unless otherwise specified.

Your code is REQUIRED to run on Python 3.11 in an environment detailed in Homework 0 under the Python Installation section. While your code may run in other environments, any points lost due to environmental issues will not be regraded.

Please make sure you don't use any libraries that are not imported for you. If such a library is used, you will get 0 pt for the coding part of the assignment.

If your code doesn't run on Gradescope, then even if it compiles on another computer, it will still be considered not running and the respective part of the assignment will receive 0 pt.

Theoretical Questions (8+11+8+13=40 pts)

Please submit your answers on Gradescope.

Q0 (8 pts): True or False Questions

Answer the following as True or False with a justification or example. Points are uniformly distributed wihtin the questions.

1. (**4 pts**) If we train a Naive Bayes classifier using infinite training data that satisfies all of its modeling assumptions, then it will achieve zero training error over these training examples.

False. This is because since Naive Bayes classifier works by estimating $P(x|y)P(y)$ through maximum likelihood estimation, an example would be: Suppose $P(x=1|y=0) = 0.1$ and $P(x=1|y=1)=0.9$, then Naive Bayes would predict $y=1$ if given $x=1$. This would result in misclassification of points where $x=1, y=0$. Thus, leading to training error.

2. (**4 pts**) The perceptron algorithm may produce different weight vectors depending on the order in which it processes the training examples.

True. This is because there could be multiple weight vectors that would result in the perceptron being able to linearly separate the training examples. Hence, a different order of training examples would result in different updating of respective weights, resulting in different magnitudes and thus eventually, producing another weight vector that could still linearly separate training examples.

Q1 (11 pts): Naive Bayes Classifier (NBC)

Consider the following classification problem, where we seek to detect the air pollution level based on two observed attributes: Location (*Rural/Suburban/Urban*) and Industrial activity (*High/Medium/Low*). We have collected 10 data points, described in the following table:

$P(L=\text{Rural}) = \frac{3}{10} = 0.3$

$P(L=\text{Urban}) = \frac{5}{10} = 0.5$

$P(L=\text{Suburban}) = \frac{2}{10} = 0.2$

Location	*Industrial activity*	*Pollution Level*
Urban	*High*	*High*
Rural	*Low*	*Low*
Urban	*Medium*	*Low*
Suburban	*Low*	*Low*
Urban	*High*	*High*
Rural	*Low*	*Low*
Urban	*Medium*	*High*
Suburban	*High*	*High*
Urban	*Low*	*Low*
Rural	*Medium*	*High*

1. **(2 pts)** Provide the equation for the probability used by a Naive Bayes classifier that predicts *Pollution level (PL)* = *Low* for an example, using *Location*(L) = *Rural* and *Industrial activity*(I) = *Medium* that is, $P(PL = Low \mid L = Rural, I = Medium)$, under the naive assumption. Please give the full equation, explicitly showing the naive assumption, with appropriate numerator and denominator. (**Note: You don't need to do calculations in this question**)

$$P(PL=\text{Low} \mid L=\text{Rural}, I=\text{Medium}) = \frac{P(L=\text{Rural}, I=\text{Medium} \mid PL=\text{Low}) \cdot P(PL=\text{Low})}{P(L=\text{Rural}, I=\text{Medium})}$$

(By Bayes Rule)

$$= \frac{P(L=\text{Rural} \mid PL=\text{Low}) \cdot P(I=\text{Medium} \mid PL=\text{Low}) \cdot P(PL=\text{Low})}{[P(L=\text{Rural}, I=\text{Medium} \mid PL=\text{Low}) \cdot P(PL=\text{Low})] + [P(L=\text{Rural}, I=\text{Medium} \mid PL=\text{High}) \cdot P(PL=\text{High})]}$$

(Numerator from naive assumption that Location and Industrial activity are conditionally independent given label Pollution Level and denominator from law of total probability)

2. **(3 pts)** For the dataset given above, compute the **NBC parameters** (probability tables) used by the Naive Bayes classifier for $P(PL)$, $P(L|PL)$ and $P(I|PL)$ (**without Laplace smoothing**). Fill in the following three tables and show necessary calculation steps.

$P(PL = High) = \frac{5}{10}$
$= 0.5$

$P(PL = High)$	$P(PL = Low)$
0.5	0.5

$P(PL = Low)$
$= \frac{5}{10} = 0.5$

Location(L)	$P(L\|PL = High)$	$P(L\|PL = Low)$
Rural	0.2	0.4
Suburban	0.2	0.2
Urban	0.6	0.4

Industrial Activity	$P(I\|PL = High)$	$P(I\|PL = Low)$
High	0.6	0
Medium	0.4	0.2
Low	0	0.8

More workings:

$P(L = Rural | PL = High) = \frac{1}{5} = 0.2$

$P(L = Suburban | PL = High) = \frac{1}{5} = 0.2$

$P(L = Urban | PL = High) = \frac{3}{5} = 0.6$

$P(L = Rural | PL = Low) = \frac{2}{5} = 0.4$

$P(L = Suburban | PL = Low) = \frac{1}{5} = 0.2$

$P(L = Urban | PL = Low) = \frac{2}{5} = 0.4$

$P(I = High | PL = High)$
$= \frac{3}{5} = 0.6$

$P(I = Medium | PL = High)$
$= \frac{2}{5} = 0.4$

$P(I = Low | PL = High)$
$= 0$

$P(I = High | PL = Low)$
$= 0$

$P(I = Medium | PL = Low)$
$= \frac{1}{5} = 0.2$

$P(I = Low | PL = Low)$
$= \frac{4}{5} = 0.8$

3. **(3 pts)** Now, compute the **same NBC parameters**, this time using Laplace smoothing (with $\alpha = 1$). Fill in the following three tables and show necessary calculation steps.

$P(PL = High)$	$P(PL = Low)$
0.5	0.5

$Location(L)$	$P(L \mid PL = High)$	$P(L \mid PL = Low)$
$Rural$	0.25	0.375
$Suburban$	0.25	0.25
$Urban$	0.5	0.375

$Industrial\ Activity$	$P(I \mid PL = High)$	$P(I \mid PL = Low)$
$High$	0.5	0.125
$Medium$	0.375	0.25
Low	0.125	0.625

More workings:

$P(L = Rural \mid PL = High) = \frac{1+1}{5+3} = 0.25$

$P(L = Suburban \mid PL = High) = \frac{1+1}{5+3} = 0.25$

$P(L = Urban \mid PC = High) = \frac{3+1}{5+3} = 0.5$

$P(L = Rural \mid PL = Low) = \frac{2+1}{5+3} = 0.375$

$P(L = Suburban \mid PL = Low) = \frac{1+1}{5+3} = 0.25$

$P(L = Urban \mid PC = Low) = \frac{2+1}{5+3} = 0.375$

$P(I = High \mid PL = High) = \frac{3+1}{5+3} = 0.5$

$P(I = Medium \mid PC = High) = \frac{2+1}{5+3} = 0.375$

$P(I = Low \mid PL = High) = \frac{1}{5+3} = 0.125$

$P(I = High \mid PL = Low) = \frac{1}{5+3} = 0.125$

$P(I = Medium \mid PL = Low) = \frac{1+1}{5+3} = 0.25$

$P(I = Low \mid PL = Low) = \frac{4+1}{5+3} = 0.625$

4. **(3 pts)** Given a new sample that is *L=Suburban* and *I=Medium* what is the probability that the air pollution level is low (*PL=Low*)? Use NBC with Laplace smoothing. If the real label is *High*, what is the 0-1 loss of your prediction?

$P(PL = Low \mid L = Suburban, I = Medium)$

$= \dfrac{P(L = Suburban, I = Medium \mid PL = Low) \cdot P(PL = Low)}{P(L = Suburban, I = Medium)}$ (By Bayes rule)

$= \dfrac{P(L = Suburban \mid PL = Low) \cdot P(I = Medium \mid PL = Low) \cdot P(PL = Low)}{[P(L = Suburban \mid PL = Low) \cdot P(I = Medium \mid PL = Low) \cdot P(PL = Low) + P(L = Suburban \mid PL = High) \cdot P(I = Medium \mid PL = High) \cdot P(PL = High)]}$

$= \dfrac{(0.25)(0.25)(0.5)}{(0.25)(0.25)(0.5) + (0.25)(0.375)(0.5)}$ (Values obtained from part (3))

$= 0.4$

$\therefore$ Probability that air pollution level is Low is < 0.5,

NBC would predict the new sample as High. Since real label is also high, the 0-1 loss of my prediction would be 0 since prediction is equal to real label.

Q2 (8 pts): Maximum Likelihood Estimation

Consider a set $\{X_1, X_2, ..., X_N\}$ of N independent and identically distributed random variables. Each of these random variables is sampled from a Bernoulli distribution X $Bernoulli(p)$.

1. **(4 pts)** Write the expression for the negative log-likelihood $\text{NLL}(X_1, \ldots, X_N) = -\log P(X_1, X_2, ..., X_N; p)$ using the probability mass function of the Bernoulli Distribution.

probability mass function of Bernoulli Distribution X Bernoulli(p):

$$P(X=x) = p^x(1-p)^{1-x}$$

For the set of N independent and identically distributed random variables $\{X_1, X_2, ..., X_n\}$, the joint likelihood is the product of all individual likelihoods:

$$L(X_1, ..., X_N; p) = \prod_{i=1}^{N} P(X_i = x_i) = \prod_{i=1}^{N} p^{x_i}(1-p)^{1-x_i}$$

$\therefore$ The log-likelihood, $LL(X_1, ..., X_N; p) = \log\left(\prod_{i=1}^{N} p^{x_i}(1-p)^{1-x_i}\right)$

$$= \sum_{i=1}^{N} \log(p^{x_i}(1-p)^{1-x_i}) \text{ (By property of log)}$$

$$= \sum_{i=1}^{N} (x_i \log p + (1-x_i)\log(1-p))$$

Hence, expression for negative log-likelihood $NLL(X_1, ..., X_N)$

$= -\log P(X_1, X_2, ..., X_N; p)$

$$= -\sum_{i=1}^{N} (x_i \log(p) + (1-x_i)\log(1-p))$$

2. **(4 pts)** Find the maximum likelihood estimate of the distribution.
Hint: Compute the gradient of the negative log-likelihood with respect to p (it should be a function of $X_1, X_2, ..., X_N$) and set it to 0. Include all derivation steps for full credits.

The negative log-likelihood (NLL), from Q1, is given by:

$$-\sum_{i=1}^{N}(x_i \log(p) + (1-x_i)\log(1-p))$$

The gradient with respect to p is thus:

$$\frac{\partial}{\partial p} NLL(X_1, ..., X_N; p) = -\sum_{i=1}^{N}\left(\frac{x_i}{p\ln 10} - \frac{1-x_i}{(1-p)\ln 10}\right) \left(\because \frac{d}{dx}\log(x) = \frac{1}{x\ln 10}\right)$$

Next, setting the derivative to zero:

$$-\sum_{i=1}^{N}\frac{x_i}{p\ln 10} - \frac{(1-x_i)}{(1-p)\ln 10} = 0$$

$$\sum_{i=1}^{N}\frac{x_i}{p\ln 10} = \sum_{i=1}^{N}\frac{1-x_i}{(1-p)\ln 10}$$

$$\sum_{i=1}^{N} x_i(1-p) = \sum_{i=1}^{N}(1-x_i)p$$

$$\sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i p = \sum_{i=1}^{N} p - \sum_{i=1}^{N} x_i p$$

$$\sum_{i=1}^{N} x_i = Np$$

Resolving for p, $p = \frac{1}{N}\sum_{i=1}^{N} x_i$

Hence, maximum likelihood estimate $= \underline{\underline{\frac{1}{N}\sum_{i=1}^{N} x_i}}$

Q3 (13 pts): Logistic Regression

In this question, we will derive the "multi-class logistic regression" algorithm. We assume the dataset D is d-dimensional (has d features) with n entries. Given a training set $\{(x^i, y^i)|i = 1, ..., n\}$ where $x^i \in \mathbb{R}^{d+1}$ is a feature vector and $y^i \in \mathbb{R}^k$ is a binary (one-hot) vector with k entries (classes). Note that in a one-hot vector, the corresponding class label is 1 and all the rest entries are 0s. For example, if the label of x is 3, then the corresponding y should look something like $[0, 0, 1, ..., 0] \in \mathbb{R}^k$.

Note that x^i is a vector of length $(d+1)$ because we pad the d features by 1 to vectorize computing the bias, that is $x^i = [1, (x^i)_0]$, where $(x^i)_0 \in D$. We want to find the parameters $\hat{w} \in \mathbb{R}^{k\times(d+1)}$ (one weight vector for each class) that maximize the likelihood for the training set, assuming a parametric model of the form,

$$p(y_c^i = 1|x^i; w) = \frac{\exp(w_c^T x^i)}{\sum_{c'} \exp(w_{c'}^T x^i)} \tag{1}$$

Note that $\frac{\exp(w_c^T x^i)}{\sum_{c'} \exp(w_{c'}^T x^i)}$ is always between 0 and 1, and $\sum_c P(y_c^i = 1|x^i; w)$ is always 1, which are desired properties of a probability distribution.

Since we know the probability sums to 1, we don't care about predicting the probability of the last (k^{th}) class, since we can calculate $p(y_k^i = 1|x^i; w)$ by:

$$P(y_k^i = 1|x^i; w) = 1 - \frac{\sum_{c'=1}^{k-1} \exp(w_{c'}^T x^i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x^i)}$$

1. **(6 points)** Derive the conditional log-likelihood for softmax regression.

Given softmax probability for a class c and a given data point x^i, denoted by equation (1), the log-likelihood for this data point x^i and its corresponding y^i is given by $\log p(y^i|x^i; w)$, where y^i is the one-hot ~~encoded~~ vector.

Then, substituting (1) into log-likelihood:

$$\log p(y^i|x^i; w) = \sum_{c=1}^{k} y_c^i \log\left(\frac{\exp(w_c^T x^i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x^i)}\right) \quad (2)$$

2. **(7 points)** Derive the gradient of the log-likelihood with respect to the c^{th} class of the weight matrix w_c, i.e., $\frac{\partial l(w)}{\partial w_c}$.

Q1. This represents the log-likelihood for a class c given a data point. We need to consider the whole dataset now and to do this, we just take the sum over all n entries. This would result in:

$$\sum_{i=1}^{n} \log p(y^i|x^i;w) = \sum_{i=1}^{n}\sum_{c=1}^{k} y_c^i \log\left(\frac{\exp(w_c^T x^i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)}\right)$$

And the above is the conditional log-likelihood for softmax regression.

Q2.

Since the log-likelihood for a single data point x_i given its corresponding y_i is: $\log p(y^i|x^i;w) = y_c^i \log\left(\frac{\exp(w_c^T x^i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)}\right)$,

Taking the derivative w.r.t w_c,

$$\frac{\partial}{\partial w_c}\log p(y^i|x^i;w) = \frac{\partial}{\partial w_c} y_c^i \log\left(\frac{\exp(w_c^T x^i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)}\right)$$

$$= \frac{\partial}{\partial w_c} y_c^i \log(\exp(w_c^T x^i)) - \frac{\partial}{\partial w_c} y_c^i \log\left(\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)\right)$$

$\therefore$

$\left(\frac{d}{dx}\log(x) = \frac{1}{x\ln 10}\right)$

$$= \frac{y_c^i}{\ln 10}\left(\frac{x^i \exp(w_c^T x^i)}{\exp(w_c^T x^i)}\right) - \frac{y_c^i}{\ln 10}\left(\frac{x_i \exp(w_c^T x^i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)}\right)$$

$(\because$ value is 1 when $c' = c$ and 0 otherwise$)$

$$= \frac{y_c^i x^i}{\ln 10} - \frac{x^i}{\ln 10}\left(y_c^i \cdot \frac{\exp(w_c^T x^i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)}\right) \quad (3)$$

Next, (3) can be simplified further:

$$\frac{y_c^i x^i}{\ln 10} - \frac{p(y_c^i = 1 | x^i; w)\, x^i}{\ln 10} \quad \left(\because p(y_c^i = 1|x^i;w) = y_c^i \cdot \frac{\exp(w_c^T x^i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x^i)}\right)$$

from (1)

$$= \frac{(y_c^i - p(y_c^i = 1|x^i;w))}{\ln 10}\, x^i$$

Then, for the entire dataset, the gradient of the log-likelihood with respect to w_c is the sum of all individual gradients for all n entries in dataset D.

Thus, $\dfrac{\partial L(w)}{\partial w_c} = \sum_{i=1}^{n} \dfrac{(y_c^i - p(y_c^i = 1 \mid x^i ; w))}{\ln 10} \cdot x^i$

Programming Part (30 + 30 = 60 Pts)

Overview

In this assignment, you will learn and implement a logistic regression classifier and also a Naive Bayes classifier.

Files

You will fill in functions in `lr.py` and `nbc.py`. After you finish, you will submit these as well as any other files mentioned in the assignment to Gradescope.

Aside from these two files, you will also find `utils.py` in the same folder. You do not need to modify these two files. `utils.py` contains some helper functions.

Packages

You will need the following packages for this assignment:

- `numpy`

If you want, you can install `tqdm` to see a progress bar when running the autograder. This is not required for this class and might not be heavily focused on in future assignments, but it is a useful package to know if you are doing any machine learning work in Python.

These packages should be installed if you have installed Anaconda. If you are not using Anaconda, you can install them using `pip` or `conda`. For example, you can run `pip install numpy` in the terminal to install `numpy`.

Note that for this assignment, you should **not** use any other packages. You may see some other packages in the skeleton code, but they are only used for testing and grading. If you use any other packages, you may lose points. If you are not sure whether you can use a package, please ask the course staff.

Evaluation

Unless otherwise specified, your code will be evaluated by an autograder on Gradescope using the same environment as detailed below. You should make sure your code runs without errors in this environment. Otherwise, you may lose points. You can submit your code to Gradescope as many times as you want. We will only grade the latest submission.

There will be two types of test cases in the autograder: public (local) and hidden. Public test cases are visible to you, and you can see the results after you submit your code. You can also run the public test cases locally by running `python <filename>.py` in the same folder as your code. Passing all public test cases **does not** guarantee you will get full credit for the assignment.

Hidden test cases, however, are not visible to you, and you will not be able to see the results until your grade is published. The final score you get for this assignment is the sum of the scores of all public and hidden test cases.

Getting Help

If you have any questions about this assignment, please contact the course staff for help, preferably during office hours. You can also post your questions on the course forum. However, please do not post any code publicly. When asking questions, you should describe your problem in words and post the relevant code snippet. Please avoid showing a screenshot of your code to TAs and ask "why my code is not working".

1 (30 pts) Logistic Regression

In this part, you will implement a logistic regression classifier. You will write your code in `lr.py`

1.1 (3 pts) LR - Constructor

Under `class LogisticRegression`, fill in the constructor so that it initializes the attributes `self.w`, which is the weight, `self.X`, and `self.y` to None. We will initialize them later in the `fit` method.

1.2 (4 pts) LR - Sigmoid

Under `class LogisticRegression`, fill in the `sigmoid` function. The function takes in a numpy array `z` and returns the sigmoid of `z`. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

1.3 (4 pts) LR - Initialize Weights

Under `class LogisticRegression`, fill in the `initialize_weights` function. The function takes in an integer `cols` representing the number of columns in the data matrix and returns a numpy array of shape `(cols, 1)` representing the initial weights. The initial weights should be initialized to 1.

1.4 (4 pts) LR - Compute Gradient

Under `class LogisticRegression`, fill in the `compute_gradient` function. The function takes in a numpy array `X` and a numpy array `y`. The function should return the gradient of the loss function with respect to the weights. The gradient is defined as:

$$\nabla_w L(w) = \frac{1}{m} X^\top (\sigma(Xw) - y)$$

where m is the number of rows in `X`, w is the weight, and σ is the sigmoid function you implemented in Q1.2.

1.5 (4 pts) LR - Fit

Under `class LogisticRegression`, fill in the `fit` function. The function takes in a numpy array X, a numpy array y, a float `lr` representing the learning rate, and an integer `epochs` representing the number of epochs.

You should first initialize the weights using the `initialize_weights` function you implemented in Q1.3. Then, you should store `X` and `y` in `self.X` and `self.y` respectively. Finally, you should run gradient descent for epochs. In each epoch, you should compute the gradient using the `compute_gradient` function you implemented in Q1.4 and update the weights using the following formula:

$$w_{i+1} = w_i - \alpha \nabla_w L(w_i)$$

where w_i is the weight at the i-th epoch and α is the learning rate.

1.6 (4 pts) LR - Predict

Under `class LogisticRegression`, fill in the `predict` function. The function takes in a numpy array X and returns a numpy array of shape `(m, 1)` where m is the number of rows in `X`. The function should return the predicted labels for `X`. The predicted labels should be 0 if the probability is less than 0.5 and 1 otherwise. The probability is defined as:

$$P(y = 1|x) = \sigma(xw)$$

where x is a row in `X`, w is the weight, and σ is the sigmoid function you implemented in Q1.2.

1.7 (4 pts) LR - Accuracy

Under `class LogisticRegression`, fill in the `accuracy` function. The function takes in a numpy array `y_pred` and a numpy array `y`. The function should return the accuracy of the predicted labels `y_pred` with respect to the true labels `y`. The accuracy is defined as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of predictions}}$$

1.8 (3 pts) LR - Overall Check

Now, you have implemented all the functions needed for logistic regression. Let's check if your implementation is correct. Run `lr.py` to train a logistic regression classifier on the training set and see the accuracy on the training set. You should get a training accuracy of at least 0.70 and a validation accuracy of at least 0.65.

There will also be a hidden test case for this question. The hidden test case will use the test set that is not provided to you. The accuracy of the hidden test set will also be used to determine your score for this question, though you are unlikely to lose points for this question if you pass all public test cases.

2 (30 pts) Naive Bayes Classifier

In this part, you will implement a Naive Bayes classifier. You will write your code in `nbc.py`.

2.1 (3 pts) NBC - Constructor

Under `class NaiveBayesClassifier`, fill in the constructor so that it takes a float `alpha` as an argument and initializes it as attribute `self.alpha`. Initialize self.`n_features`, `self.class_labels`, `self.class_probabilities`, and `self.feature_probabilities` to `None`. We will assign them later.

2.2 (4 pts) NBC - Prior Probability

Under `class NaiveBayesClassifier`, fill in the `compute_class_probabilities` function. The function takes in a numpy array `y_train` and returns a dictionary `class_probabilities` where the keys are the class labels and the values are the prior probabilities of the class labels. The prior probability of a class label is defined as:

$$P(y = c) = \frac{\text{Number of samples with label c}}{\text{Number of samples}}$$

where c is a class label.

2.3 (4 pts) NBC - Feature Probability

Under `class NaiveBayesClassifier`, fill in the `compute_feature_probabilities` function. The function takes in a numpy array `X_j_train` and a numpy array `y_train`. The function should return a dictionary `feature_probabilities` where the keys are the tuple (`feature_value`, `class_label`) and the values are the probabilities of the feature values given the class labels. Use Laplace smoothing with parameter `self.alpha`.

2.4 (4 pts) NBC - Fit

Under `class NaiveBayesClassifier`, fill in the `fit` function. The function takes in a numpy array `X_train` and a numpy array `y_train`. The function should store `X_train` and `y_train` in `self.X_train` and `self.y_train` respectively. Then, the function should compute the class (prior) probabilities of the class labels and feature probabilities of the feature values given the class labels. Finally, the function should store the class probabilities and feature probabilities in `self.class_probabilities` and `self.feature_probabilities` respectively.

2.5 (4 pts) NBC - Predict Probability

Under `class NaiveBayesClassifier`, fill in the `predict_probabilities` function. The function takes in a numpy array `X_test` and returns a numpy array of shape (m, 1) where m is the number of rows in `X_test`. The function should return the predicted probabilities for X.

2.6 (4 pts) NBC - Predict

Under `class NaiveBayesClassifier`, fill in the `predict` function. The function takes in a dictionary of probabilities and returns a numpy array of shape `(m, 1)` where m is the number of data points. The function should return the predicted labels. The predicted labels should be the class labels with the highest probability.

2.7 (4 pts) NBC - Evaluate

Under `class NaiveBayesClassifier`, fill in the evaluate function. The function takes in a numpy array `y_test` and a numpy array probabilities and returns a tuple of two floats (`zero_one_loss`, `squared_loss`). The function should return the zero-one loss and the customized squared loss of the predicted labels with respect to the true labels. The zero-one loss is defined as:

$$\text{Zero-one loss} = \frac{\text{Number of incorrect predictions}}{\text{Number of predictions}}$$

The customized squared loss is defined as:

$$\text{Customized squared loss} = \frac{1}{m}\sum_{i=1}^{m}(1 - p_i)^2$$

where m is the number of data points, y_i is the true label of the i-th data point, and $\hat{y}_i$ is the predicted label of the i-th data point.

2.8 (3 pts) NBC - Overall Check

Now, you have implemented all the functions needed for the Naive Bayes classifier. Let's check if your implementation is correct. Run `nbc.py` to train a Naive Bayes classifier on the training set and see the accuracy of the training set. You should get a zero-one loss of at most 0.3 and a squared loss of at most 0.25.

There will also be a hidden test case for this question. The hidden test case will use the test set that is not provided to you. The zero-one loss and squared loss on the hidden test set will also be used to determine your score for this question, though you are unlikely to lose points for this question if you pass all public test cases.

Submission

You will submit the following files to Gradescope:

- `lr.py`
- `nbc.py`