EECS 545 WN24 Machine Learning
Homework #2

Due date: 11:55 PM, Tuesday, Feb 13, 2024

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your solutions and code independently. Do not share your solution with anyone else in the class. If you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2024/eecs545` with a reference to the specific question in the subject line (e.g., Homework 2, Q3(b): can we assume XXX conditions?).

Submission Instruction: You should submit both **writeup** and **source code**. We may inspect your source code submission visually and rerun the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **writeup** to **Gradescope** (`https://www.gradescope.com/`)
  - Your writeup should contain your answers to **all** questions (typeset or hand-written), except for the implementation-only questions (marked with **(Autograder)**).
  - **For each subquestion, please select the associated pages from the pdf file in Gradescope. The graders are not required to look at pages other than the ones selected, and this may lead to some penalty when grading**.
  - Your writeup should be self-contained and self-explanatory. You should include the plots and figures in your writeup whenever asked, even when they are generated from the ipython notebook.
  - Please typeset (with LaTeX) or hand-write your solutions legibly. **Hand-writing that is difficult to read may lead to penalties**. If you prefer hand-writing, please use scanners or mobile scanning apps that provide shading corrections and projective transformations for better legibility; you **should *not*** just take photos and submit them without any image processing.
  - For all math derivations, you **should** provide detailed explanations as much as possible. If not, you may not be able to get a full credit if there are any mistakes or logical jumps.
- Submit **source code** to **Autograder** (`https://autograder.io/`)
  - Each ipython notebook file (i.e., `*.ipynb`) will walk you through the implementation task, producing the outputs and plots for *all* subproblems. You are required to write code on its matched python file (`*.py`). For instance, if you are working on `logistic_regression.ipynb`, you are required to write code on `logistic_regression.py`. Note that the outputs of your source code must match with your **writeup**.
  - We will read the data file in the `data` directory (i.e., `data/*.npy`) **from the same (current) working directory**: for example, `X_train = np.load('data/q1x.npy')`.
  - When you want to evaluate and test your implementation, please submit the `*.py` and `*.ipynb` files to Autograder for grading your implementations in the middle or after finish everything. You can partially test some of the files anytime, but please note that this also reduces your daily submission quota. You can submit your code up to **three** times per day.

- The autograder will give you information to help you troubleshoot your code. The following is the current list of error codes (will be updated accordingly if we encounter more common student implementation errors).

  0: Success
  1: The program exited with an unknown test failure.
  2: The program exited with an un-handled exception (usually a runtime error)
  3-5: Some other runtime error indicating an issue with the autograder code.
  40: NotImplementedError
  50: An uncategorized test failure different from the following error codes. Most likely indicates wrong outputs, but this may also include other assertion failures.
  51: There is an incorrect dtype on some output tensor.
  52: There is an incorrect shape on some output tensor.
  53: Some output tensors are not equal, or too different given the tolerance.
  54: Some output tensors are not equal with difference 1.0. Possibly due to rounding issues.
  55: The value of some input tensor was changed.
  60: The model's performance or converged loss is not good enough.
  61: The model might have diverged (e.g. encountered NaN or inf).
  62-63: The model has converged to a bad likelihood.

- Your program should run under an environment with following libraries:
  - Python 3.11 [1]
  - NumPy (for implementations of algorithms)
  - Matplotlib (for plots)

  Please do not use any other library unless otherwise instructed (no third-party libraries other than numpy and matplotlib are allowed). You should be able to load the data and execute your code on Autograder with the environment described above, but in your local working environment you might need to install them via `pip install numpy matplotlib`.

- For this assignment, you will need to submit the following files:
  - `logistic_regression.py`
  - `logistic_regression.ipynb`
  - `softmax_regression.py`
  - `softmax_regression.ipynb`
  - `naive_bayes_spam.py`
  - `naive_bayes_spam.ipynb`

  Do not change the filename for the code and ipython notebook file because the Autograder may not find your submission. Please do not submit any other files except `*.py` and `*.ipynb` to Autograder.

- When you are done, please upload your work to Autograder (sign in with your UMich account). To receive the full credit, your code must run and terminate without error (i.e., with exit code 0) in the Autograder. Keep all the cell outputs in your notebook files (`*.ipynb`). We strongly recommend you run **Kernel → Restart Kernel and Run All Cells** (in the Jupyter Lab menu) before submitting your code to Autograder.

Credits

Some problems were adopted from Stanford CS229 and Bishop PRML.

[1] We recommend using Miniconda (`https://docs.conda.io/en/latest/miniconda.html`). You can create `eecs545` environment with Python 3.11 as `conda create --name eecs545 python=3.11`. It is fine to use other python distributions and versions as long as they are supported, but we will run your program with Python 3.11 on Autograder.

Change Log

- rev0 (2024/1/30 1PM): Initial Release

1 [25 points] Logistic Regression

Consider the log-likelihood function for logistic regression:

$$\ell(\mathbf{w}) = \sum_{i=1}^{N} y^{(i)} \log h(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h(\mathbf{x}^{(i)})),$$

where $h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \dfrac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}$.

(a) **[5 points]** Find the Hessian H of $\ell(\mathbf{w})$.

(b) **[6 points]** Show that H is negative semi-definite and thus ℓ is concave and has no local maxima other than the global one. That is, show that

$$\mathbf{z}^\top H \mathbf{z} \leq 0$$

for any vector $\mathbf{z}$. [*Hint: You might want to start by showing the fact that* $\sum_i \sum_j z_i x_i x_j z_j = (\mathbf{x}^\top \mathbf{z})^2$. *Note that* $(\mathbf{x}^\top \mathbf{z})^2 \geq 0$.]

(c) **[2 points]** Using the H you calculated in part (a), write down the update rule implied by Newton's method for optimizing $\ell(\mathbf{w})$. [*Hint: it could be a single-line equation:* $\mathbf{w} = YOUR\_ANSWER$.]

(d) **[8 points]** (Autograder) Now use the update rule in (c) (and not a library function) to implement Newton's method and apply it to binary classification problem, following the guide in `logistic_regression.ipynb`. Your `ipynb` file **SHOULD** include all the outputs.

(e) **[2 points]** What are the coefficients $\mathbf{w}$, including the intercept term, resulting from your code? Please provide your answer in your **writeup**.

(f) **[2 points]** Please share the final plot from `logistic_regression.ipynb` in your **writeup**.

2 [27 points] Softmax Regression via Gradient Ascent

Gradient ascent is an algorithm used to find parameters that maximize a certain expression (contrary to gradient descent, which is used to minimize an expression). For some function $f(\mathbf{w})$, gradient ascent finds $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} f(\mathbf{w})$ according to the following pseudo-code:

Algorithm 1 Gradient Ascent

1: $\mathbf{w}^* \leftarrow$ *random*
2: **repeat**
3: $\quad\quad \mathbf{w}^* \leftarrow \mathbf{w}^* + \alpha \nabla_{\mathbf{w}} f(\mathbf{w}^*)$
4: **until** *convergence*
5: **return** $\mathbf{w}^*$

Softmax regression is a multiclass classification algorithm. Given a labeled dataset $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), ..., (\mathbf{x}^{(N)}, y^{(N)})\}$, where $y^{(i)} \in \{1, 2, ..., K\}$ (total K classes), softmax regression computes the probability that an example $\mathbf{x}$ belongs to a class k:

$$p(y = k|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \phi(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \phi(\mathbf{x}))}$$

where $\mathbf{w}_k$ is a weight vector for class k. The above expression is over-parametrized, meaning that there is more than one unique $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K\}$ that gives identical probability measures for $p(y = k|\mathbf{x}, \mathbf{w})$. An unique solution can be obtained using only $K-1$ weight vectors $\mathbf{w} = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{K-1}\}$ and fixing $\mathbf{w}_K = 0$:

$$p(y = k|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \phi(\mathbf{x}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^\top \phi(\mathbf{x}))}; \;\; \forall k = \{1, 2, ..., K-1\}$$

$$p(y = K|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^\top \phi(\mathbf{x}))}$$

We define the likelihood of the ith training example $p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$ as:

$$p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \prod_{k=1}^{K} \Big[p(y^{(i)} = k|\mathbf{x}^{(i)}, \mathbf{w})\Big]^{\mathbb{I}\{y^{(i)}=k\}}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function. We define the likelihood as:

$$L(\mathbf{w}) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \prod_{i=1}^{N} \prod_{k=1}^{K} \Big[p(y^{(i)} = k|\mathbf{x}^{(i)}, \mathbf{w})\Big]^{\mathbb{I}\{y^{(i)}=k\}}$$

Finally, we define the log-likelihood as:

$$l(\mathbf{w}) = \log L(\mathbf{w}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \log(\Big[p(y^{(i)} = k|\mathbf{x}^{(i)}, \mathbf{w})\Big]^{\mathbb{I}\{y^{(i)}=k\}})$$

(a) **[11 points]** Derive the gradient ascent update rule for the log-likelihood of the training data. In other words, derive the expression $\nabla_{\mathbf{w}_m} l(\mathbf{w})$ for $m = 1, ..., K-1$. Show that:

$$\nabla_{\mathbf{w}_m} l(\mathbf{w}) = \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \left[\mathbb{I}\{y^{(i)} = m\} - \frac{\exp(\mathbf{w}_m^\top \phi(\mathbf{x}^{(i)}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^\top \phi(\mathbf{x}^{(i)}))}\right]$$

$$= \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \Big[\mathbb{I}\{y^{(i)} = m\} - p(y^{(i)} = m | \mathbf{x}^{(i)}, \mathbf{w}) \Big]$$

Remember that in this notation, $\mathbf{w}_k$ is a weight vector for class k, **not** the kth element of $\mathbf{w}$.

[Hints: $\log a^b = b \log(a)$*. Further, it helps to consider the two cases separately; a case for* $y^{(i)} = k = m$*, and another for* $y^{(i)} = k \neq m$*, which is equivalent to using Kronecker delta* δ_{km}*].*

(b) **[14 points]** (Autograder) Using the gradient computed in part (a), implement gradient ascent for softmax regression, following the guide in `softmax_regression.ipynb`. You are required to implement your code in `softmax_regression.py`. Make sure to include all the outputs in your `softmax_regression.ipynb` file. Recall that softmax regression classifies an example $\mathbf{x}$ as:

$$y = \underset{y'}{\operatorname{argmax}}\, p(y' | \mathbf{x}, \mathbf{w})$$

(c) **[2 points]** When you train your classifier on the training data in (b), what is the accuracy on the test data? Please report your accuracy in your **writeup**.

3 [25 points] Gaussian Discriminate Analysis

Suppose we are given a dataset $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, ..., N\}$ consisting of N independent examples, where $\mathbf{x}^{(i)} \in \mathbb{R}^M$ are M-dimensional vectors, and $y^{(i)} \in \{0, 1\}$. We will model the joint distribution of $(\mathbf{x}^{(i)}, y^{(i)})$ using:

$$p(y^{(i)}) = \phi^{y^{(i)}}(1-\phi)^{1-y^{(i)}}$$

$$p(\mathbf{x}^{(i)}|y^{(i)}=0) = \frac{1}{(2\pi)^{\frac{M}{2}}|\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_0)^\top \Sigma^{-1}(\mathbf{x}^{(i)}-\mu_0))$$

$$p(\mathbf{x}^{(i)}|y^{(i)}=1) = \frac{1}{(2\pi)^{\frac{M}{2}}|\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_1)^\top \Sigma^{-1}(\mathbf{x}^{(i)}-\mu_1))$$

Here, the parameters of our model are ϕ, Σ, μ_0, and μ_1. (Note that while there are two different mean vectors μ_0 and μ_1, there is only one covariance matrix Σ.)

(a) **[6 points]** Suppose we have already fit ϕ, Σ, μ_0, and μ_1, and now want to make a prediction at some new query point $\mathbf{x}$. Show that the posterior distribution of the label (y) at $\mathbf{x}$ takes the form of a logistic function, and can be written as

$$p(y=1|\mathbf{x}; \phi, \Sigma, \mu_0, \mu_1) = \frac{1}{1+\exp(-\mathbf{w}^\top \widehat{\mathbf{x}})}$$

where $\widehat{\mathbf{x}}$ to be $M+1$ dimensional vectors ($\widehat{\mathbf{x}}^{(i)} \in \mathbb{R}^{M+1}$) by adding an extra coordinate $\widehat{\mathbf{x}}_0 = 1$ from the original $\mathbf{x}$ and $\mathbf{w}$ is a function of ϕ, Σ, μ_0, and μ_1.

(b) **[14 points]** The maximum likelihood estimates of ϕ, μ_0, and μ_1 are given by

$$\phi = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}=1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}=0\}\mathbf{x}^{(i)}}{\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}=0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}=1\}\mathbf{x}^{(i)}}{\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}=1\}}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function. The log-likelihood of the data is

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{N} p(\mathbf{x}^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$$

$$= \log \prod_{i=1}^{N} p(\mathbf{x}^{(i)}|y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi)$$

By maximizing ℓ with respect to the three parameters (ϕ, μ_0, and μ_1), **prove that** the maximum likelihood estimates of ϕ, μ_0, and μ_1 are indeed the ones described above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_0 and μ_1 above are non-zero.)

(c) **[5 points]** When M (the dimension of $\mathbf{x}^{(i)}$) is 1, then $\Sigma = \left[\sigma^2\right]$ becomes a scalar, and its determinant is $|\Sigma| = \sigma^2$. By maximizing ℓ in (b) with respect to Σ, prove the maximum likelihood estimate for Σ would become

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \mu_{y^{(i)}})(\mathbf{x}^{(i)} - \mu_{y^{(i)}})^\top.$$

4 [23 points] Naive Bayes for Classifying SPAM

(a) **[6 points]** Naive Bayes with Bayesian Smoothing

Recall that Naive Bayes can be solved with MLE, in which we count the occurences of each feature (or word). Adding Laplace smoothing, we get:

$$P(C_i) = \phi_i = \frac{N^{C_i}}{\sum_{i'} N^{C_{i'}}} \tag{1}$$

$$P(x_j|C_i) = \mu_j^i = \frac{N_j^{C_i} + \alpha}{\sum_{j'} N_{j'}^{C_i} + \alpha K} \tag{2}$$

where K is the number of classes and M is the dimension of $\mathbf{x}$ (total number of features or words). $N_j^{C_i}$ is the count of the occurrences of x_j with class C_i. $\alpha > 0$ is the Laplace smoothing hyperparameter.

Show that Laplace smoothing is equivalent to solving the MAP estimate of Naive Bayes, where we have a prior on the values of $\boldsymbol{\mu}$ which follow a symmetric Dirichlet distribution:

$$P(\boldsymbol{\mu}) = \frac{1}{Z} \prod_{i=1}^{K} \prod_{j=1}^{M} \left(\mu_j^i\right)^{\alpha} \tag{3}$$

where Z is some normalizing constant.

[Hint: You may use the Naive Bayes likelihood and MLE derivations from lecture without proof.]

(b) In this subproblem, we will use the naive Bayes algorithm to build a SPAM classifier which we covered in our class. Our classifier will distinguish between "real" (non-spam) emails and spam emails. For this problem, we obtained a set of spam emails and a set of non-spam newsgroup messages. Using only the subject line and body of each message, the classifier will learn to distinguish between the two groups. In this problem, we're going to call the features *tokens*.

i. **[9 points]** (Autograder) Implement a naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing. `naive_bayes_spam.ipynb` will work you through implementing this classifier. In this task, we will train your parameters with the training dataset **MATRIX.TRAIN**. Then, use these parameters to classify the test dataset **MATRIX.TEST** and compute the accuracy using the **evaluate** function. Implement your code in `naive_bayes_spam.py` and submit your `naive_bayes_spam.py` implementation with `naive_bayes_spam.ipynb`. Please include all the outputs in your `naive_bayes_spam.ipynb` file.

ii. **[2 points]** Some tokens may be particularly indicative of an email being spam or non-spam. One way to measure how indicative a token i is for the SPAM class by looking at:

$$\log\left(\frac{p(x_j = i|y = 1)}{p(x_j = i|y = 0)}\right) = \log\left(\frac{P(\text{token}_i|\text{email is SPAM})}{P(\text{token}_i|\text{email is NOTSPAM})}\right)$$

Using the parameters you obtained in part (a), find the 5 tokens that are most indicative of the SPAM class (i.e., 5 tokens that have the highest positive value on the measure above). What are the first five tokens from your implementation. Please provide them in your **writeup**.

iii. **[2 points]** Repeat part (a), but with different naive Bayes classifiers each trained with different training sets **MATRIX.TRAIN.***. Evaluate the classifiers with **MATRIX.TEST** and report the classification accuracy for each classifier in your **writeup**.

iv. **[2 points]** Please provide the final training data size-accuracy plot from part (c) in your **writeup**.

v. **[2 points]** Which training set size gives you the best classification accuracy? Please provide your own analysis in your **writeup**.