CMSC 35400 / STAT 37710

Spring 2020
Homework 5

You must clearly indicate where your solutions to individual subproblems are in gradescope. If you force the graders to do this for you, points will be deducted from your total.

1. **(Poisson Naive Bayes)** In this task we will use the Naive Bayes model for binary classification. Let $\mathcal{Y} = \{0, 1\}$ be the set of labels and $\mathcal{X} = \mathbb{N}^d$ a d-dimensional features space ($\mathcal{N} = \{0, 1, 2, \ldots\}$). You are given a training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n labeled examples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

   a) Is the Naive Bayes model a generative or a discriminative model? Justify your answer.

   SOLUTION: The Naive Bayes model is a generative model because it models the joint data-generating distribution $P(X, Y)$.

   b) Let λ be a positive scalar, and assume that $z_1, \ldots, z_m \in \mathbb{N}$ are m iid observations of a λ-Poisson distributed random variable. Find the maximum likelihood estimator for λ in this model. (Hint: A λ-Poisson distributed random variable Z takes values $k \in \mathbb{N}$ with probability $P(Z = k) = e^{-\lambda} \frac{\lambda^k}{k!}$.)

   SOLUTION: The MLE for Poisson(λ) distribution is the empirical mean:

$$\hat{\lambda} = \frac{1}{m} \sum_{j=1}^{m} z_m.$$

   To verify this, write down the likelihood function:

$$L(\lambda; z_1, \ldots, z_m) = \prod_{j=1}^{m} e^{-\lambda} \frac{\lambda^{z_j}}{z_j!}$$

   The log-likelihood function is

$$\ell(\lambda; z_1, \ldots, z_m) = -m\lambda - \sum_{j=1}^{m} \log(z_j!) + \log \lambda \sum_{j=1}^{m} z_j$$

   Setting the gradient w.r.t. λ to 0, and we hence get the MLE estimate.

   c) Let's train a Poisson Naive Bayes classifier using maximum likelihood estimation. Define appropriate parameters $p_0, p_1 \in [0, 1]$, and vectors $\lambda_0, \lambda_1 \in R^d$, and write down the joint distribution $P(X, Y)$ of the resulting model. (Note that the following should be satisfied for the parameters: $p_0 + p_1 = 1$, and λ_0, λ_1 are vectors with non-negative components.)

   SOLUTION: Let n be the total number of data points, $n_1 = \sum_{i=1}^{n} y_i$ the number of times '1' was observed, and $n_0 = n - n_1$ the number of '0' accordingly. The Naive Bayes model in our case is

$$p(x, y) = p(y) \prod_{j=1}^{d} p(x_j \mid y).$$

The MLE for $p(y) = \text{Bernoulli}(\theta)$ is simply the empirical frequency $p_y = \frac{n_y}{n}$. Similarly the MLE for a Poisson $P(\lambda)$ distribution is just the empirical mean (see previous step). Hence we estimate $\lambda_{y,i} = \frac{\sum_{i=1}^{n} x_{i,j} 1\{y_i=y\}}{n_y}$. The resulting distribution is

$$p(x, y) = p_y \prod_{j=1}^{d} e^{-\lambda_{y,j}} \frac{\lambda_{y,j}^{x_j}}{x_j!}$$

d) Now, we want to use our trained model from b) to minimize the misclassification probability of a new observation $x \in \mathcal{X}$, i.e., $y_{\text{pred}} = \arg\max_{y \in \mathcal{Y}} P(y \mid X = x)$. Show that the predicted label y_{pred} for x is determined by a hyperplane, i.e., that $y_{\text{pred}} = [a^\top x \geq b]$ for some $a \in \mathbb{R}^d, b \in \mathbb{R}$.

SOLUTION: The joint distribution from the Naive Bayes model is

$$p(x, y) = p_y \prod_{j=1}^{d} e^{-\lambda_{y,j}} \frac{\lambda_{y,j}^{x_j}}{x_j!}$$

We are interested in the decision boundary $p(y = 0 \mid x) = p(y = 1 \mid x)$. We rewrite this as

$$\begin{aligned}
P(y = 0 \mid x) &= p(y = 1 \mid x) \\
\Leftrightarrow p(x, 0) &= p(x, 1) \\
\Leftrightarrow p_0 \prod_{j=1}^{d} e^{-\lambda_{0,j}} \frac{\lambda_{0,j}^{x_j}}{x_j!} &= p_1 \prod_{j=1}^{d} e^{-\lambda_{1,j}} \frac{\lambda_{1,j}^{x_j}}{x_j!} \\
\Leftrightarrow \log\left(\frac{p_0}{p_1}\right) + \sum_{j=1}^{d} -\lambda_{0,j} + \log(\lambda_{0,j}) x_j &= \sum_{j=1}^{d} -\lambda_{1,j} + \log(\lambda_{1,j}) x_j
\end{aligned}$$

From the last equation the claim follows, i.e. the decision is determined by the hyperplane

$$\log\left(\frac{p_0}{p_1}\right) + \sum_{j=1}^{d} (\lambda_{1,j} - \lambda_{0,j}) + \log\left(\frac{\lambda_{0,j}}{\lambda_{1,j}}\right) x_j = 0$$

e) Instead of simply predicting the most likely label, one can define a cost function $c : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, such that $c(y_{\text{pred}}, y_{\text{true}})$ is the cost of predicting y_{pred} given that the true label is y_{true}. Define the Bayes optimal predictor for a cost function $c(\cdot, \cdot)$, with respect to a distribution $P(X, Y)$ as

$$y_{\text{Bayes}} = \underset{y \in \mathcal{Y}}{\arg\min} \mathbb{E}_Y [c(Y, y) \mid X = x].$$

Write down a cost function such that the corresponding Bayes optimal predictor for this cost coincides with a predictor that minimizes the misclassification probability, i.e., $y_{\text{pred}} = \arg\max_{y \in \mathcal{Y}} P(y \mid X = x)$.

SOLUTION: The cost function cerrosponds to the 0/1 loss:

$$c(y_{\text{pred}}, y_{\text{true}}) = 1\{y_{\text{true}} \neq y_{\text{pred}}\}.$$

2. **(Multiclass logistic regression)** The posterior probabilities for mulitclass logistic regression can be given as a softmax transformation of hyperplanes, such that:

$$P(y = k \mid X = x) = \frac{\exp(a_k^\top x)}{\sum_j \exp(a_j^\top x)}$$

If we consider the use of maximum likelihood to determine the parameters a_k, we can take the negative logarithm of the likelihood function to obtain the *cross-entropy* error function for multiclass logistic regression:

$$E(a_1, \ldots, a_K) = -\ln\left(\prod_{n=1}^{N}\prod_{k=1}^{K} P(y = k \mid X = x_n)^{t_{nk}}\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln P(y = k \mid X = x_n)$$

where $t_{nk} = 1\{\text{labelOf}(x_n) = k\}$.

Show that the gradient of the error function can be stated as given below

$$\nabla_{a_k} E(a_1, \ldots, a_K) = \sum_{n=1}^{N} \left[P(y = k \mid X = x_n) - t_{nk}\right] x_n$$

SOLUTION: Define $d_k = a_k^\top x$. The posterior probabilities are given as

$$P(y = k \mid X = x) = \frac{\exp(a_k^\top x)}{\sum_j \exp(a_j^\top x)} = y_k(x)$$

First, we compute the derivatives of y_k with respect to all d_j's:

$$\frac{\partial y_k}{\partial d_j} = y_k(1\{k = j\} - y_j)$$

This holds because if $j \neq k$, we have

$$\frac{\partial y_k}{\partial d_j} = \frac{-\exp(d_k)\exp(d_j)}{\left[\sum_j \exp(d_j)\right]^2} = -y_k \cdot y_j$$

and if $j = k$,

$$\frac{\partial y_k}{\partial d_j} = \frac{\exp(d_k)\sum_j \exp(d_j) - \exp(d_k)\exp(d_k)}{\left[\sum_j \exp(d_j)\right]^2} = y_k \cdot (1 - y_k)$$

Next, we compute the partial derivative of the summands of $E(a_1, \ldots, a_K)$

$$\frac{\partial t_{nk} \ln y_k(x_n)}{\partial a_j} = \frac{\partial [t_{nk} \ln y_k(x_n)]}{\partial [y_k(x_n)]} \frac{\partial [y_k(x_n)]}{\partial d_j} \frac{\partial d_j}{\partial a_j}$$

Let $y_{nk} = y_k(x_n)$, we simplify to

$$\frac{\partial t_{nk} \ln y_k(x_n)}{\partial a_j} = t_{nk} \frac{1}{y_{nk}} y_{nk}(1\{k = j\} - y_{nj})x_n = t_{nk}(1\{k = j\} - y_{nj})x_n$$

Then, using the fact that $\sum_{k=1}^{K} t_{nk} = 1$, and $y_{nk} = y_k(x_n) = P(y = k \mid X = x_n)$, we get

$$\begin{aligned}
\nabla_{a_j} E(a_1, \ldots, a_K) &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk}(1\{k = j\} - y_{nj})x_n \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} y_{nj} x_n - \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} 1\{k = j\} x_n \\
&= \sum_{n=1}^{N} \left(\sum_{k=1}^{K} t_{nk} \right) y_{nj} x_n - \sum_{n=1}^{N} t_{nj} x_n \\
&= \sum_{n=1}^{N} [y_{nj} - t_{nj}] \, x_n \\
&= \sum_{n=1}^{N} [P(y = j \mid X = x_n) - t_{nj}] \, x_n
\end{aligned}$$

Therefore, for any k, it holds that $\nabla_{a_k} E(a_1, \ldots, a_K) = \sum_{n=1}^{N} [P(y = k \mid X = x_n) - t_{nk}] \, x_n$.

3. **(Programming exercise: Regularized logistic regression)** In this exercise, We will use regularized logistic regression to solve a classification task. Consider the following dataset:

```python
#python
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification

X, y = make_classification(n_features=2,
                           n_redundant=0,
                           n_informative=2,
                           random_state=1,
                           n_clusters_per_class=1)

rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
```

```
linearly_separable = (X, y)

X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=.4, random_state=42)
```

a) Implement regularized Logistic Regression (LR) using gradient descent (or stochastic gradient descent). Use constant step size $\eta_t = 0.01$, and regularization constant, $\lambda = 0.1$. Choose an appropriate threshold value as stopping criteria to decide if the weights are converged.

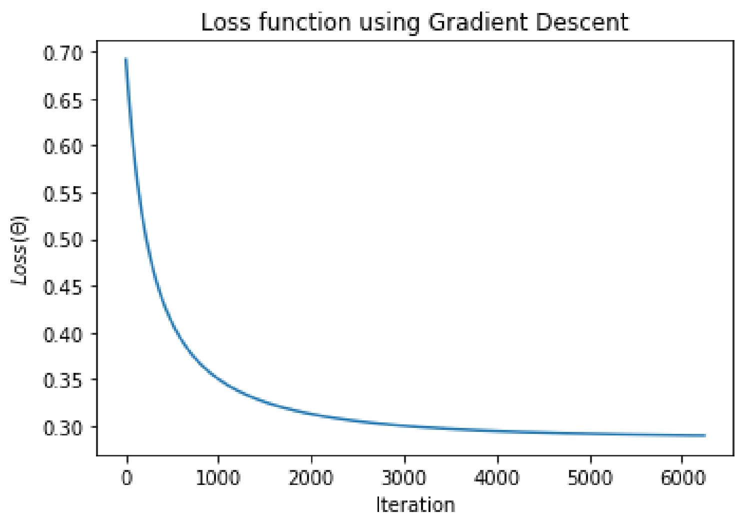
Note that you are **not** allowed to use existing machine learning packages for logistic regression (e.g. `sklearn.linear_model.LogisticRegression`) for this exercise.

```
# Initialize fitting parameters
initial_theta = np.zeros((X.shape[1], 1))
# Set regularization parameter lambda and learning rate eta
Lambda = 0.1
Eta = 0.01

def lossFunction(theta, X, y ,Lambda):
#
# IMPLEMENT THE LOSS AND GRADIENT FUNCTION
# OF REGULARIZED LOGISTIC REGRESSION
#
#
    return loss, grad

def gradientDescent(X,y,theta,eta,Lambda,tolerance):
#
# IMPLEMENT THE (STOCHASTIC) GRADIENT DESCENT ALGORITHM
# USING THE lossFunction DEFINED ABOVE
#
#
    return theta
```

SOLUTION: see attached soluction file `hw5_logistic_reg_sol.ipynb`

b) Plot negative log likelihood with respect to iterations needed to converge, and report the amount of time spent training the classifier with the corresponding stopping criteria.
SOLUTION: see attached soluction file `hw5_logistic_reg_sol.ipynb`.

c) Train the classifiers you constructed in step (a) on the training set `X_train, y_train`.
Plot the decision boundary, and report training accuracy (i.e., accuracy on `X_train, y_train`), testing accuracy (i.e., `X_test, y_test`).



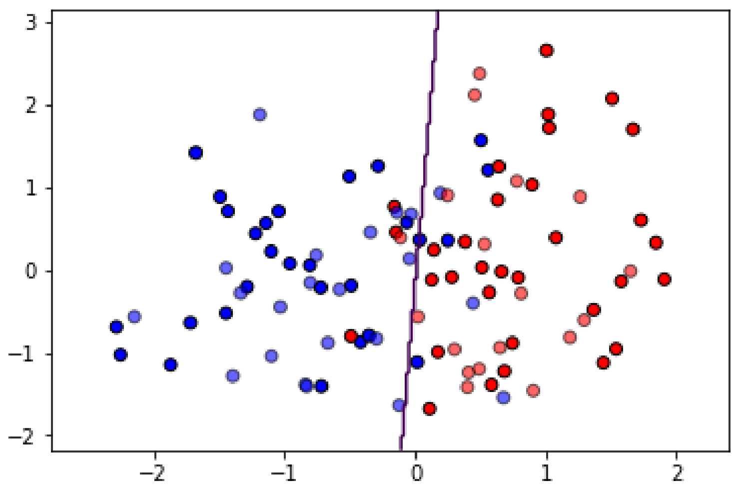
To help you get started, you may consider to add your code to the following snippet to plot the decision boundary of your trained regularized logistic regression model:

```
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# plot the dataset
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])

# Plot the training points
plt.scatter(X_train[:, 0], X_train[:, 1],c=y_train,
            cmap=cm_bright,edgecolors='k')
# and test points
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test,
            cmap=cm_bright, alpha=0.6,edgecolors='k')

# ADD YOUR CODE TO PLOT THE DECISION BOUNDARY
#
#
```

SOLUTION: see attached soluction file `hw5_logistic_reg_sol.ipynb`.

`Train Accuracy: 86.66666666666667 %   Test Accuracy: 90.0 %`.

The results may vary as random number generator may not perform consistently on different platforms even with the same random state.


3
2
1
0
−1
−2
−2
−1
0
1
2