

TTIC 31020: Introduction to Statistical Machine Learning
Autumn 2015

Problem Set #4

Out: November 21, 2016

Due: Thursday December 1, 11:59pm

Instructions

Please read these instructions carefully and follow them precisely. Feel free to ask the instructor if anything is unclear!

How and what to submit? Please submit your solutions electronically via Canvas. Please submit two files:

1. A PDF file with the written component of your solution including derivations, explanations, etc. You can create this PDF in any way you want: typeset the solution in L^AT_EX (recommended), type it in Word or a similar program and convert/export to PDF, or even hand write the solution (legibly!) and scan it to PDF. Please name this document `<firstname-lastname>-sol1.pdf`.
2. The empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures) in a Jupyter notebook file.

In addition, you will need to submit your predictions on the handwritten digit recognition tasks to Kaggle, as described below, according to the competition rules. Please contact the TAs with any questions of difficulties.

Late submissions: there will be a penalty of 25 points for any solution submitted within 24 hours past the deadline. No submissions will be accepted past then.

What is the required level of detail? When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked

to plot something, please include the figure as well as the code used to plot it (and clearly explain in the `README` what the relevant files are). If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers). When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important.

When submitting code, please make sure it's reasonably documented, and describe succinctly what is done in the relevant bits of code in the notebook.

1 Regularization

We will consider the L_2 norm regularization for squared loss in regression:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \sum_{i=1}^N (y_i - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i))^2 + \lambda \sum_{j=1}^d w_j^2 \right\}, \quad (1)$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a $d+1$ -dimensional feature vector, including the constant term $\phi_0 \equiv 1$ which we do not regularize, and N is the size of the training set.

Problem 1 [25 points]

Show that the objective in (1) is equivalent to

$$\begin{aligned} \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \sum_{i=1}^N (y_i - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i))^2 \right\} \\ \text{subject to } \sum_{j=1}^d w_j^2 \leq \tau \end{aligned} \quad (2)$$

for appropriate τ , which may depend in some way on the data and on λ . In other words, if you find \mathbf{w}^* according to (1), then find \mathbf{w}^* according to (2) with the appropriate τ , the two values of \mathbf{w}^* will be the same.

End of problem 1

Advice: It may be helpful to consider the solution of the constrained problem in (2) using Lagrange multipliers.

2 Generative Models

In this section, we will work with two generative models: Naïve Bayes and Mixture Models. We will explore their performance on two datasets you're already familiar with: MNIST (the large train portion from pset2 now with binarized features) for digit classification and Consumer Reviews (from pset3) for sentiment analysis.

There will be Kaggle competitions to compare the performance of Mixture Models applied separately to MNIST and Consumer Reviews, while there will be no Kaggle component for the implemented Naïve Bayes model. The same data, however, is used for both models and must be accessed through Kaggle.

2.1 Naïve Bayes

In this problem you will implement a Naïve Bayes (NB) classifier to be applied for binary (sentiment) and multi-class (digit) classification.

We have not discussed Naïve Bayes classifiers in class; below is the brief overview of this approach.

A generative model, as we have seen in class, assigns the label according to a discriminant rule

$$\hat{y}(\mathbf{x}) = \underset{c}{\operatorname{argmax}} \{ \log p(\mathbf{x} | y = c) + \log p(y = c) \}. \quad (3)$$

The model of class-conditional densities $p(\mathbf{x} | y = c)$ for each class c is typically parametric and fit separately to examples from each class. The model for the priors $p(c)$ is usually simply based on counting – relative frequency of the classes in the training data. In the common scenario of balanced data where all C classes are represented with the same number of examples, the priors boil down to the uniform $p(c) = 1/C$ and so can be ignored since they don't contribute to the decision in (3).

A commonly used approach to regularization of such models is to limit, or penalize, the complexity of the class-conditionals. In the case of Gaussian $p(\mathbf{x} | y)$, we can consider restricting the covariance matrices for the class-specific Gaussian models. One such restriction is to limit the covariance

matrix to be diagonal for each class. If $\mathbf{x} \in \mathbb{R}^d$, then this means

$$\Sigma_c = \begin{bmatrix} \sigma_{c,1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{c,2}^2 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \sigma_{c,d}^2 \end{bmatrix} \quad (4)$$

Under this restriction, the form of the Gaussian distribution for class c can be simplified:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \Sigma_c) = \frac{1}{(2\pi)^{d/2} |\Sigma_c|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_c)^T \Sigma_c^{-1} (\mathbf{x} - \boldsymbol{\mu}_c) \right) \quad (5)$$

$$= \frac{1}{(2\pi)^{d/2} \prod_{j=1}^d \sigma_{c,j}} \exp \left(-\frac{1}{2} \sum_{j=1}^d (x_j - \mu_{c,j})^2 / \sigma_{c,j}^2 \right) \quad (6)$$

$$= \prod_{j=1}^d \frac{1}{(2\pi)^{1/2} \sigma_{c,j}} \exp \left(-\frac{1}{2} (x_j - \mu_{c,j})^2 / \sigma_{c,j}^2 \right) \quad (7)$$

where $\mu_{c,j}$ is the j -th coordinate of $\boldsymbol{\mu}_c$. So, under the diagonal covariance assumption, the joint distribution of features (dimensions) of \mathbf{x} given class c is simply the product of distributions of individual features given that class. By definition, this means that the features of \mathbf{x} are independent conditioned on the class.

This is the Naïve Bayes model. In its general form, the NB model for d -dimensional input examples \mathbf{x} can be written as

$$\log p(\mathbf{x} | y = c) = \sum_{j=1}^d \log p(x_j | y = c), \quad (8)$$

where $p(x_j | y)$ specifies the class-conditional model for a one-dimensional (scalar) value of the j -th feature of \mathbf{x} given the class.

It's called "naïve" since this conditional independence assumption is somewhat simplistic. However, it often works well in practice, both because of the regularizing effect of the independence assumption (think how many parameters does the general Gaussian model have vs. the NB Gaussian model) and because estimating class-conditional density is much easier computationally if it needs to be done per feature rather than jointly for many features.

In the following problem, we will use a Bernoulli distribution for each binary feature $p(x_j | y)$, where j ranges over the indices of the pixels in the

input images for MNIST, or over the indices of word/bigram occurrence features for the text analysis. You will need to write the code estimating the parameters of the Bernoulli distributions, and of course the NB classifier. Assume for the purpose of this assignment that the priors are equal for both/all classes, and can be ignored.

Accessible through the below Kaggle competition link is the dataset labeled `datasets-ps4.h5` which is an hdf5 file similar to the one used in `pset2`. Within the datafile the following are accessible, note, however, that functions are provided within the `utils.py` file and notebook to load the data once the hdf5 file is in your working directory:

```
sentiment_analysis/train/data
sentiment_analysis/train/labels
sentiment_analysis/val/data
sentiment_analysis/val/labels
sentiment_analysis/kaggle/data
```

```
mnist/train/data
mnist/train/labels
mnist/val/data
mnist/val/labels
mnist/kaggle/data
```

Here are the kaggle competition links:

- <https://kaggle.com/join/mixturemodelConsumerReviews>
- <https://kaggle.com/join/mixturemodelMNIST>

With regards to the Consumer Reviews data, your NB classifier will need to use binary features based on the occurrence of bigram features (as used in `pset3`). With regards to the MNIST dataset, the features are (differently from in `pset2`) **binarized** pixel values 0 or 1. Code is provided to load the train/val data using the `load_experiment()` function in `utils.py`, and the kaggle data is loaded using the `load_data()` function.

Problem 2 [20 points]

Use the sentiment and digit classification data described above for training (i.e. for estimating the parameters). Construct an NB classifier using ML estimate for Bernoulli parameters.

Please evaluate your trained models on the corresponding validation sets (for MNIST and Consumer Reviews), then report the accuracy and confusion matrix in the ipython notebook. How might you change the parameter estimation to improve this fairly “naïve” model?

End of problem 2

3 Mixture models

In this part of the problem set we will derive and implement the EM algorithm for a discrete space, in which the observations are d -dimensional binary vectors (containing either 0 or 1). We will model distributions in this space as a mixture of multivariate Bernoulli distributions. That is,

$$p(\mathbf{x} | \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{l=1}^k \pi_l p(\mathbf{x} | \boldsymbol{\theta}_l), \quad (9)$$

where $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_k]$ are the parameters of the k components, and the mixing probabilities $\boldsymbol{\pi} = [\pi_1, \dots, \pi_k]^T$ are subject to $\sum_l \pi_l = 1$.

The l -th component of the mixture is parametrized by a d -dimensional vector $\boldsymbol{\theta}_l = [\theta_{l1}, \dots, \theta_{ld}]^T$. The value of θ_{lj} is the probability of 1 in the j -th coordinate in a vector drawn from this distribution. Since the dimensions are assumed to be independent, given $\boldsymbol{\theta}_l$, the conditional distribution of \mathbf{x} under this component is

$$p(\mathbf{x} | \boldsymbol{\theta}_l) = \prod_{j=1}^d p(x_j | \theta_{lj}) = \prod_{j=1}^d \theta_{lj}^{x_j} (1 - \theta_{lj})^{1-x_j}. \quad (10)$$

The hidden variables here are, just in the Gaussian mixture case, the identities of the component that generated each observation. We will denote the hidden variable associated with \mathbf{x}_i by z_i . The EM with the model proceeds as follows. The E-step consists of computing the posterior of the hidden variables, $\gamma_{i,c} = p(z_i = c | \mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\pi})$. In the M-step, we need to update the estimates of $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$. These updates can be derived in a way similar to the derivation for the Gaussian mixture.

Problem 3 [15 points]

Write the exact expression for the posterior probability $\gamma_{i,c} = p(z_i = c | \mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\pi})$

in terms of $\mathbf{x}_1, \dots, \mathbf{x}_N$ and the elements of $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$.

End of problem 3

Problem 4 [15 points]

Derive the M-step updates for the multivariate Bernoulli mixture, using given $\gamma_{i,c}$ values from the preceding E-step.

End of problem 4

This model could be applicable, for instance, in modeling distributions of consumer reviews using binary features. In the following problem, we will apply it to the sentiment analysis task we have encountered in an earlier problem set, as well as a multi-class classification objective in MNIST digit classification.

One method of using EM in a discriminative setting is to train a separate mixture model for each class. That is, in the context of digit classification, one EM would only see images of the digit 0 and a different EM would see each of the other digits 1–9. It is common for generative models to be trained separately for each label like this

$$p(\mathbf{x} | y = c, \boldsymbol{\theta}_c, \boldsymbol{\pi}_c) = \sum_{l=1}^k \pi_{cl} p(\mathbf{x} | \boldsymbol{\theta}_{cl}), \quad \forall c \quad (11)$$

Once these are trained, a discriminative model emerges naturally by selecting the class with maximum likelihood for a given sample

$$\hat{y} = \underset{c}{\operatorname{argmax}} p(\mathbf{x} | y = c, \boldsymbol{\theta}_c, \boldsymbol{\pi}_c). \quad (12)$$

A common simplification, which you may use if you want, is to assume that one component will dominate the likelihood and thus compare the likelihoods separately for each component

$$\hat{y} = \underset{c}{\operatorname{argmax}} \max_l p(\mathbf{x} | \boldsymbol{\theta}_{cl}). \quad (13)$$

The mixing weights are in this case ignored. Note that this is equivalent to viewing all the mixtures models as a single hierarchical mixture model and then selecting the leaf component with the maximum likelihood.

Problem 5 [25 points]

Implement a classifier for the aforementioned tasks based on multiple Bernoulli mixture models as described above. A new Kaggle competition has been set up, and the data as well as further descriptions are available at this link (note: this is the same link as above copied again for your convenience):

- <https://kaggle.com/join/mixturemodelConsumerReviews>
- <https://kaggle.com/join/mixturemodelMNIST>

A few practical considerations when implementing EM:

- The EM algorithm is sensitive to initialization. You may find that randomly setting your weights will not give good results. A simple method that works well in practice is to randomly assign samples to a component and then calculating the corresponding means of those clusters. This may seem strange, since all components will start very close to the population mean. However, it works well in practice.
- When your means (θ) get close to 0 or 1, it can cause overfitting and instability. One way of solving this problem is by deciding on an $\epsilon \in (0, \frac{1}{2})$ and each time you update your means make sure that $\theta \in [\epsilon, 1 - \epsilon]$. If they lie outside of this range, you simply snap them to the closest boundary value. You will have to decide an appropriate value for ϵ .
- You will need calculations of the general form

$$f(\mathbf{x}) = \log \sum_i \exp(x_i)$$

If the values of \mathbf{x} have large absolute values, you will run into numerical underflow or overflow as $\exp(x_i)$ approach zero or infinity. In our case, we are worried about underflow, since we are dealing with events of very low probability. A common trick is to find an appropriate constant C and realize that the following is mathematically (but not numerically) equivalent

$$f(\mathbf{x}) = C + \log \sum_i \exp(x_i - C),$$

where typically $C = \max(\mathbf{x})$. There is a function in Python that does this: `scipy.misc.logsumexp`

If you find it difficult to debug your EM, a suggestion is to run it on MNIST first. This way, you will be able to visually inspect your components. If it works correctly, your components should look like prototypical digits.

End of problem 5