HW2. Gaussian Discriminant Analysis

Jongmin Lee (jmlee@ai.kaist.ac.kr)

March 22, 2016

1 Introduction

In this assignment, you will design Gaussian Discriminant Analysis classifiers on two image data sets which are same as before (i.e. handwritten digit images and face images).

2 Numpy and Anaconda

In this assignment, we will use NumPy(`http://www.numpy.org/`), which makes it easier to handle many matrix-vector operations (addition, multiplication, and so on). If you don't have NumPy yet, we strongly recommend you to install it through Anaconda (`https://www.continuum.io/downloads`). Anaconda is not only free and cross-platform, but everything including NumPy and other scientific packages is precompiled, so it will be installed very quickly.

- Anaconda includes Python installer itself, so you don't need to have Python in advance.
- Don't forget to use **Python 2.7** when scripting your code.

3 Project Instruction

3.1 Gaussian Discriminant Analysis (GDA)

The generative learning algorithm that we'll deal with this time is Gaussian discriminant analysis (GDA). In this model, we'll assume that $P(\vec{x}|y)$ is distributed according to a multivariate Gaussian distribution. The model is:

$$\begin{aligned} y &\sim \text{Categorical}(y|\pi_1, ..., \pi_C) \\ \vec{x}|y = c &\sim \mathcal{N}(\vec{x}|\vec{\mu}_c, \Sigma_c) \end{aligned}$$

3.1.1 Linear Discriminant Analysis (LDA)

In linear discriminant analysis (LDA), we assume that covariance matrices are shared among different classes (i.e. $\Sigma_1 =, ..., = \Sigma_C = \Sigma$). To classify a datum, we derive the posterior distribution on y given $\vec{x}$ using Bayes

rule:

$$
\begin{aligned}
P(y=c|\vec{x}) &= \frac{P(\vec{x}|y=c)P(y=c)}{\sum_{c'} P(\vec{x}|y=c')P(y=c')} \\
&= \frac{\mathcal{N}(\vec{x}|\vec{\mu}_c, \Sigma)\pi_c}{\sum_{c'} \mathcal{N}(\vec{x}|\vec{\mu}_{c'}, \Sigma)\pi_{c'}} \\
&= \frac{\frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(\vec{x}-\vec{\mu}_c)^\top \Sigma^{-1}(\vec{x}-\vec{\mu}_c))\pi_c}{\sum_{c'} \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(\vec{x}-\vec{\mu}_{c'})^\top \Sigma^{-1}(\vec{x}-\vec{\mu}_{c'}))\pi_{c'}} \\
&= \frac{\exp(\vec{\mu}_c^\top \Sigma^{-1}\vec{x} - \frac{1}{2}\vec{\mu}_c^\top \Sigma^{-1}\vec{\mu}_c + \log \pi_c)}{\sum_{c'} \exp(\vec{\mu}_{c'}^\top \Sigma^{-1}\vec{x} - \frac{1}{2}\vec{\mu}_{c'}^\top \Sigma^{-1}\vec{\mu}_{c'} + \log \pi_{c'})} \\
&= \frac{\exp(\vec{\beta}_c^\top \vec{x} + \gamma_c)}{\sum_{c'} \exp(\vec{\beta}_{c'}^\top \vec{x} + \gamma_{c'})}
\end{aligned}
$$

, where $\vec{\beta}_c = \Sigma^{-1}\vec{\mu}_c$ and $\gamma_c = -\frac{1}{2}\vec{\mu}_c^\top \Sigma^{-1}\vec{\mu}_c + \log \pi_c$. The decision rule is given by

$$y = \arg\max_c P(y=c|\vec{x})$$

Note that $\exp(-\frac{1}{2}\vec{x}^\top \Sigma^{-1}\vec{x})$ term cancel out and decision boundaries become linear since Σ's are shared among classes.

3.1.2 Quadratic Discriminant Analysis (QDA)

In quadratic discriminant analysis (QDA), we don't assume any covariance matrix sharing, which results in quadratic decision boundaries. We derive the posterior distribution y given $\vec{x}$ using Bayes rule:

$$
\begin{aligned}
P(y=c|\vec{x}) &= \frac{P(\vec{x}|y=c)P(y=c)}{\sum_{c'} P(\vec{x}|y=c')P(y=c')} \\
&= \frac{\mathcal{N}(\vec{x}|\vec{\mu}_c, \Sigma_c)\pi_c}{\sum_{c'} \mathcal{N}(\vec{x}|\vec{\mu}_{c'}, \Sigma_{c'})\pi_{c'}} \\
&= \frac{\frac{1}{(2\pi)^{D/2}|\Sigma_c|^{1/2}} \exp(-\frac{1}{2}(\vec{x}-\vec{\mu}_c)^\top \Sigma_c^{-1}(\vec{x}-\vec{\mu}_c))\pi_c}{\sum_{c'} \frac{1}{(2\pi)^{D/2}|\Sigma_{c'}|^{1/2}} \exp(-\frac{1}{2}(\vec{x}-\vec{\mu}_{c'})^\top \Sigma_{c'}^{-1}(\vec{x}-\vec{\mu}_{c'}))\pi_{c'}}
\end{aligned}
$$

The decision rule is given by

$$y = \arg\max_c P(y=c|\vec{x})$$

3.1.3 Parameter Estimation

Our LDA and QDA models have several parameters to estimate (i.e. π_c, $\vec{\mu}_c$ and Σ_c). We'll use **maximum likelihood estimates (MLE)**. Likelihood and log likelihood functions are given by:

$$
\begin{aligned}
P(X|\boldsymbol{\pi}, \vec{\boldsymbol{\mu}}, \boldsymbol{\Sigma}) &= \prod_{n=1}^{N}\prod_{c=1}^{C} \pi_c^{\mathbb{I}(y_n=c)} \mathcal{N}(\vec{x}_n|\mu_c, \Sigma_c)^{\mathbb{I}(y_n=c)} \\
\log P(X|\boldsymbol{\pi}, \vec{\boldsymbol{\mu}}, \boldsymbol{\Sigma}) &= \sum_{n=1}^{N}\sum_{c=1}^{C} \mathbb{I}(y_n=c)\log \pi_c + \mathbb{I}(y_n=c)\log \mathcal{N}(\vec{x}_n|\mu_c, \Sigma_c) \\
&= \sum_{n=1}^{N}\sum_{c=1}^{C} \left[\mathbb{I}(y_n=c)\log \pi_c + \mathbb{I}(y_n=c)\left(-\frac{1}{2}\log 2\pi + \frac{1}{2}\log|\Sigma_c^{-1}| - \frac{1}{2}(\vec{x}_n-\vec{\mu}_c)^\top \Sigma_c^{-1}(\vec{x}_n-\vec{\mu}_c)\right)\right]
\end{aligned}
$$

We can compute MLEs simply by taking partial derivatives with respect to π_c, $\vec{\mu}_c$ and Σ_c^{-1} respectively and set them to zero. (Computing $\hat{\pi}_c$ is a little bit exceptional because there is a constraint $\sum_c \pi_c = 1$ and we will use Lagrange multiplier for this.)

3.1.4 MLE Derivation: $\hat{\pi}_c$

For example, we can compute MLE for π_c as follows. First, we define Lagrangian $\mathcal{L}$ since we have to maximize log-likelihood while satisfying the constraint $\sum_{c=1}^{C} \pi_c = 1$.

$$\begin{aligned}\mathcal{L} &= \log P(X|\boldsymbol{\pi}, \vec{\boldsymbol{\mu}}, \boldsymbol{\Sigma}) + \lambda \left(1 - \sum_{c=1}^{C} \pi_c\right) \\ &= \sum_{n=1}^{N}\sum_{c=1}^{C}\left[\mathbb{I}(y_n = c)\log \pi_c + \mathbb{I}(y_n = c)\left(-\frac{1}{2}\log 2\pi + \frac{1}{2}\log|\Sigma_c^{-1}| - \frac{1}{2}(\vec{x}_n - \vec{\mu}_c)^\top \Sigma_c^{-1}(\vec{x}_n - \vec{\mu}_c)\right)\right] + \lambda\left(1 - \sum_{c=1}^{C}\pi_c\right)\end{aligned}$$

The next step is to take derivatives $\mathcal{L}$ with respect to π_c and λ and set them to zero.

$$\frac{\partial \mathcal{L}}{\partial \pi_c} = \sum_{n=1}^{N} \frac{\mathbb{I}(y_n = c)}{\pi_c} - \lambda = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 1 - \sum_{c=1}^{C} \pi_c = 0$$

This gives us $\lambda = N$ and the MLE solution:

$$\boxed{\hat{\pi}_c = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(y_n = c)}$$

3.1.5 MLE Derivation: $\hat{\vec{\mu}}_c$ and $\hat{\Sigma}_c$

How to compute $\hat{\vec{\mu}}_c$ and $\hat{\Sigma}_c$? It's your turn! It is also a part of this assignment :)

3.2 What to Do

You will fill in portions of **gaussianDiscriminantAnalysis.py**. Specifically, you have to code the following functions:

- *trainAneTune*
- *calculateLogJointProbabilities*

In *trainAndTune* function, estimate class prior probabilities and conditional densities from the training data (i.e. compute the maximum likelihood estimates for each class) for LDA and QDA respectively. Evaluate both LDA and QDA and choose the model that gives higher accuracy on the validation data.
The method *calculateLogJointProbabilities* uses class prior probability and conditional probability densities constructed by *trainAndTune* to compute log posterior probability for each class y given a feature vector. Since *trainAndTune* chooses classification method according to validation accuracy, if LDA had higher accuracy than QDA on validation dataset during training, *calculateLogJointProbabilities* method will compute posterior probability through stored LDA model. If QDA was better, QDA will be used for prediction. The comments of the method describe the data structures of the input and output.

- To simplify the problem, we serve dimensionality reduced real-valued feature vectors obtained by PCA as input parameters. We will learn PCA(Principle Component Analysis) later in the class.

- Do **NOT** use LDA and QDA function which are implemented in other scientific packages(i.e. scikit-learn) directly.

3.3 What to Submit

Please submit **gaussianDiscriminantAnalysis.py** file only. Any late submissions will not be accepted.

3.4 How to Run the Code

To try out the classification pipeline, run **dataClassifier.py** from the command line. This will classify the digit data using the default classifier (mostFrequent) which blindly classifies every example with the most frequent label.

```
python dataClassifier.py
```

To activate the Gaussian Discriminant Analysis classifier, use -c GDA:

```
python dataClassifier.py -c GDA
```

To run on the face recognition dataset with different training data size, use -d faces and -t numTraining

```
python dataClassifier.py -c GDA -d faces -t 300
```