Lab#4: Generalized Linear Model and Generative Learning

Prepared by: Teeradaj Racharak (r.teeradaj@gmail.com)

Generalized linear model

Problem 1 [TensorFlow].

Let's modify the logistic regression classifier we implemented in Scenario 2 of Lab 3

$$p(y = i \mid \boldsymbol{x}; \boldsymbol{\Theta}) = \phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} = \frac{e^{\boldsymbol{\theta}_i^T \boldsymbol{x}}}{\sum_{j=1}^{k} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}}$$

to use softmax regression instead of binary logistic regression. Noted that the hypothesis function in softmax is as follows (see slide#17 of Lecture 4.0):

Instead of hard-coding the above function, TensorFlow also provides off-the-shelf functions for computing this; so, you may consider to use it in your implementation. See https://www.tensorflow.org/api_docs/python/tf/nn/softmax for more detail.

Generative learning algorithm

TL;DR It's ok. you may jump directly to Problem 2.

In linear regression and logistic regression, we model both as a conditional distribution of y given x as follows:

Linear regression: $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$

Logistic regression: $p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = h_\theta(\boldsymbol{x}) = g(\boldsymbol{\theta}^T \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$

Algorithms that models $p(y \mid \boldsymbol{x})$ directly from the training set are called 'discriminative' algorithms.

As we have seen in the lecture note, there is alternative way for the same problem. Let's consider the same binary classification problem where we want to distinguish two classes *viz.* class A ($y = 1$) and class B ($y = 0$) based on the same features. Now, we take all examples of label A and try to learn the features and build a model for class A. Then, we take all the examples labeled B and try to learn its feature and build a separate model for class B. Finally, to classify a new element, we match it against each model and see which one fits better (*i.e.* generate high value for probability). In this approach, we try to model $p(\boldsymbol{x} \mid y)$ and $p(y)$ as oppose to $p(y \mid \boldsymbol{x})$ where we did earlier. This is called 'generative learning' algorithms.

Once we can learn the models $p(y)$ and $p(\boldsymbol{x}\,|\,y)$ using the training set, we use Bayes rule to derive $p(y\,|\,\boldsymbol{x})$ as follows:

$$p(y\,|\,\boldsymbol{x}) = \frac{p(\boldsymbol{x}\,|\,y)\cdot p(y)}{p(\boldsymbol{x})}$$

where $p(\boldsymbol{x}) = p(\boldsymbol{x}\,|\,y=1)\cdot p(y=1) + p(\boldsymbol{x}\,|\,y=0)\cdot p(y=0)$.

From the lecture note, we know that if we are calculating $p(y\,|\,\boldsymbol{x})$ in order to make a prediction, we don't need $p(\boldsymbol{x})$ as:

$$\underset{y}{\operatorname{argmax}}\, p(y\,|\,\boldsymbol{x}) = \underset{y}{\operatorname{argmax}}\, \frac{p(\boldsymbol{x}\,|\,y)\cdot p(y)}{p(\boldsymbol{x})} = \underset{y}{\operatorname{argmax}}\, p(\boldsymbol{x}\,|\,y)\cdot p(y)$$

Gaussian discriminant analysis (GDA)

If we have a classification problem in which the input features are continuous random variable, we can use GDA. If this is the case, we assume that $p(\boldsymbol{x}\,|\,y)$ is distributed according to multivariate normal distribution and $p(y)$ is distributed according to Bernoulli. So, the model is:

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

$$p(\boldsymbol{x}\,|\,y=0) = \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\mu_0)^T\Sigma^{-1}(\boldsymbol{x}-\mu_0)}$$

$$p(\boldsymbol{x}\,|\,y=1) = \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\mu_1)^T\Sigma^{-1}(\boldsymbol{x}-\mu_1)}$$

Here, the parameters of the model are ϕ, μ_0, μ_1, and Σ. And, n is the dimension of the distribution. Noted that there are two separate mean vectors μ_0 and μ_1 (for each class); however, we assume that the covariance matrix Σ is common for both classes.

As we did in linear regression and logistic regression, we need to define the log likelihood function l and maximize it w.r.t. the model parameters. We can find the maximize likelihood parameters as follows:

$$\begin{aligned} l(\phi,\mu_0,\mu_1,\Sigma) &= \log \Pi_{i=1}^{m} p(\boldsymbol{x}^{(i)}, y^{(i)}; \phi,\mu_0,\mu_1,\Sigma) \\ &= \log \Pi_{i=1}^{m} p(x^{(i)}\,|\,y^{(i)}; \phi,\mu_0,\mu_1,\Sigma)\cdot p(y^{(i)};\phi) \end{aligned}$$

Let's rewrite the log likelihood becomes more generic w.r.t. k classes. This means we have $k = 2$ for the current function. The generic log likelihood is rewritten as follows:

$$l(\phi,\mu_k,\Sigma) = \log \Pi_{i=1}^{m} p(\boldsymbol{x}\,|\,y;\phi,\mu_k,\Sigma)\cdot p(y;\phi)$$

$$= \log \Pi_{i=1}^{m} \Big(\frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\mu_k)^T \Sigma^{-1} (\boldsymbol{x}-\mu_k)} \Big) \cdot \phi^y (1-\phi)^{(1-y)}$$

$$= \Sigma_{i=1}^{m} \Big[-\frac{n}{2} \log 2\pi - \frac{1}{2} \log |\Sigma| - \frac{1}{2} \big(\boldsymbol{x}-\mu_k\big)^T \Sigma^{-1} (\boldsymbol{x}-\mu_k)$$

$$+ y \log \phi + (1-y) \log(1-\phi) \Big] \qquad (1)$$

Now, it's time to take the partial derivatives w.r.t. each parameter and set them to 0 in order to find the maximum likelihood of that parameter.

To solve for parameter ϕ, we derive as follows:

$$\frac{\partial}{\partial \phi} l(\phi, \mu_k, \Sigma) = 0 \iff \Sigma_{i=1}^{m} \Big[\frac{y^{(i)}}{\phi} - \frac{1-y^{(i)}}{1-\phi} \Big] = 0 \text{ (by Equation 1)}$$

$$\implies \Sigma_{i=1}^{m} \big[y^{(i)}(1-\phi) - \phi(1-y^{(i)}) \big] = 0$$

$$\implies \Sigma_{i=1}^{m} \big[y^{(i)} - y^{(i)}\phi - \phi + \phi y^{(i)} \big] = 0$$

$$\implies \Sigma_{i=1}^{m} \big[y^{(i)} - \phi \big] = 0$$

$$\implies \Sigma_{i=1}^{m} y^{(i)} - m\phi = 0 \; (\because \Sigma_{i=1}^{m} \phi = \phi \Sigma_{i=1}^{m} 1 = \phi m)$$

$$\implies \phi = \frac{1}{m} \Sigma_{i=1}^{m} y^{(i)}$$

If $y \in \{0,1\}$ (*i.e.* binary classification), we can rewrite the above as follows:

$$\phi = \frac{1}{m} \Sigma_{i=1}^{m} 1\{y^{(i)} = 1\} \qquad (2)$$

where $1\{\cdot\}$ represents a function the returns 1 if $\cdot$ be true and returns 0 if otherwise.

To solve for parameters μ_0, μ_1, we derive as follows:

$$\frac{\partial}{\partial \mu_k} l(\phi, \mu_k, \Sigma) = 0 \iff \Sigma_{i=1}^{m} \Big[-\frac{1}{2} \frac{\partial}{\partial \mu_k} \big((\boldsymbol{x}^{(i)} - \mu_k)^T \Sigma^{-1} (\boldsymbol{x}^{(i)} - \mu_k) \big) \Big] = 0 \; (*)$$

We know $\frac{\partial}{\partial \alpha} \alpha^T A x = 2\alpha^T x$ as A is symmetric and doesn't depend on x (*cf.* Theorem 1 in the appendix). Putting $\alpha = (x_k^{(i)} - \mu_k)$; also, by the chain rule

$$\frac{\partial l}{\partial \mu_k} = \frac{\partial l}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial \mu_k}$$

and

$$\frac{\partial \alpha}{\partial \mu_k} = \begin{cases} -1, & \text{if } (i \,|\, y^{(i)}) = k \\ 0, & \text{otherwise} \end{cases} \iff 1\{y^{(i)} = k\}; \tag{3}$$

We now derive as follows:

$$(*) \implies -\frac{1}{2} \cdot 2\Sigma_{i=1}^{m} \Big[(x^{(i)} - \mu_k)^T \Sigma^{-1} \frac{\partial \alpha}{\partial \mu_k} \Big] = 0$$

$$\implies \Sigma_{i=1}^{m} x_k^{(i)} \frac{\partial \alpha}{\partial \mu_k} - \Sigma_{i=1}^{m} \mu_k \frac{\partial \alpha}{\partial \mu_k} = 0$$

$$\implies \mu_k = \frac{\Sigma_{i=1}^{m} 1\{y^{(i)} = k\} x^{(i)}}{\Sigma_{i=1}^{m} 1\{y^{(i)} = k\}} \text{ (by Equation 3)}$$

Putting $k = 0, 1$; then, we have:

$$\mu_0 = \frac{\Sigma_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\Sigma_{i=1}^{m} 1\{y^{(i)} = 0\}} \tag{4}$$

and

$$\mu_1 = \frac{\Sigma_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\Sigma_{i=1}^{m} 1\{y^{(i)} = 1\}} \tag{5}$$

To solve for parameter Σ, we derive as follows:

$$\frac{\partial}{\partial \mu_k} l(\phi, \mu_k, \Sigma) = 0 \iff \Sigma_{i=1}^{m} \Big[-\frac{1}{2} \Big(\frac{\partial}{\partial \Sigma} \log(|\Sigma|) \Big) - \frac{1}{2} \frac{\partial}{\partial \Sigma} \Big((x^{(i)} - \mu_k)^T \Sigma^{-1} (x^{(i)} - \mu_k) \Big) \Big] = 0$$

$$\implies \Sigma_{i=1}^{m} \Big[-\frac{1}{2} \Sigma^{-T} - \frac{1}{2} \Big(-\Sigma^{-T} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T \Sigma^{-T} \Big) \Big] = 0$$

$$\Big(\because \frac{\partial \log |X|}{\partial X} = X^{-T} \text{ and } \frac{\partial}{\partial X} \big[a^T X^{-1} b \big] = -X^{-T} a b^T X^{-T} \Big)$$

Taking out $-\frac{1}{2}\Sigma^{-T}$ common from the above equation, we have:

$$\implies \Sigma_{i=1}^{m} \big[1 - \Sigma^{-T} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T \big] = 0$$

$$\implies m - \Sigma_{i=1}^{m} \Sigma^{-T} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T = 0$$

$$\implies m\Sigma = \Sigma_{i=1}^{m} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T$$

$$\implies \Sigma = \frac{1}{m} \Sigma_{i=1}^{m} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T, \text{ where } k = 1\{y^{(i)} = 1\} \tag{6}$$

To conclude, we have shown all the maximum likelihood parameters of GDA, *viz.* Equation 2, 4, 5, and 6. Maximizing likelihood parameters of Naive Bayes classifier can also be done in the similar fashion !

Now, we are ready to practice using GDA classifier and Naive Bayes classier.

In the following, we will practice using another famous machine learning libraries named 'scikit-learn' to implement GDA and Naive Bayes classifiers. As reported in AI Index 2018 Annual Report (see Figure 1), scikit-learn is the second popular machine learning learning reached in Github. So, you should also know about it ! By the way, what is AI index? Check out this link: https://aiindex.org/

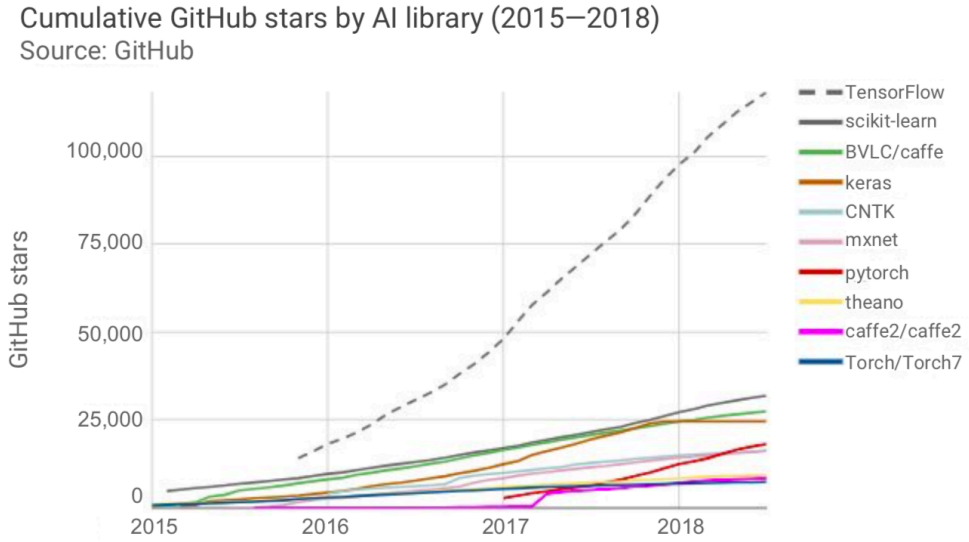

Figure 1 Cumulative Github stars by AI library (2015 - 2018)

Problem 2 [GDA].

Now, we will try using GDA to explore something interesting inside the Pokemon dataset. Our goal is to 'classify the type of Pokemon' based on its stat value. Well, what is Pokemon and what is stat values? We compile the list of stats as follows:

- name: The English name of the Pokemon
- japanese_name: The Original Japanese name of the Pokemon
- pokedex_number: The entry number of the Pokemon in the National Pokedex
- percentage_male: The percentage of the species that are male. Blank if the Pokemon is genderless.
- type1: The Primary Type of the Pokemon
- type2: The Secondary Type of the Pokemon
- classification: The Classification of the Pokemon as described by the Sun and Moon Pokedex

- height_m: Height of the Pokemon in metres
- weight_kg: The Weight of the Pokemon in kilograms
- capture_rate: Capture Rate of the Pokemon
- base_egg_steps: The number of steps required to hatch an egg of the Pokemon
- abilities: A stringified list of abilities that the Pokemon is capable of having
- experience_growth: The Experience Growth of the Pokemon
- base_happiness: Base Happiness of the Pokemon
- against_?: Eighteen features that denote the amount of damage taken against an attack of a particular type
- hp: The Base HP of the Pokemon
- attack: The Base Attack of the Pokemon
- defense: The Base Defense of the Pokemon
- sp_attack: The Base Special Attack of the Pokemon
- sp_defense: The Base Special Defense of the Pokemon
- speed: The Base Speed of the Pokemon
- generation: The numbered generation which the Pokemon was first introduced
- is_legendary: Denotes if the Pokemon is legendary.

Applying GDA to the dataset can tell us how linearly separable the dataset is.

It is worth mentioning that, in GDA model, if $\Sigma_1 = \Sigma_2 = \Sigma$, then this is alternatively called 'linear discriminant analysis'; otherwise, it will be called 'quadratic discriminant analysis'. In scikit-learn, this kind of GDA model is also implemented as the python class 'LinearDiscriminantAnalysis'.

In this exercise, we will classify type of Pokemon based on the following stat:

- sp_attack
- sp_defense
- attack
- defense
- speed
- hp

We will consider only Pokemon with a single type *i.e.* no dual types allowed. This requirement can be posed by coding the following in pandas:

```
dataframe = pokemon[pokemon['type2'].isnull()].loc[
    :, ['sp_attack', 'sp_defense', 'attack', 'defense', 'speed', 'hp', 'type1']
]
```

After you have trained the model, try to convince yourself if types of Pokemon can be linearly distinguishable based on only stat values.

(Comment: One way to convince ourselves about the result is to visualize it. How should we do that since there are many stat values? Ones may guess that we need to compress dimensions so that the original dimensions are reduced to 2D. Otherwise, we cannot plot (well, 3D is also possible). Linear discriminant analysis is also useful for this task. See https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html#sklearn.discriminant_analysis.LinearDiscriminantAnalysis.fit_transform for more detail)

Problem 3 [Naive Bayes]

In this exercise, we will create a spam classifier (*i.e.* Ham or Spam?) using Naive Bayes with scikit-learn. We retrieve a dataset remotely with the following code:

```
url = 'https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv'
sms_dataset = pd.read_table(url, header=None, names=['label', 'message'])
```

Our goal is to classify if an sms is a spam based on a certain occurrence of words.

It is worth mentioning that we cannot directly feed a sequence of symbols into the algorithm. Instead, we need to convert each sequence of symbols into numerical feature vectors with a fixed size. This process is normally called 'Vectorizing'. In scikit-learn, we can use 'CountVectorizer' to convert text into a matrix of token counts. See https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html for more detail. The vectorized matrix is sometimes called the 'bag of words'. If you take a look inside it, you can see that each text is described by word occurrences while completely ignoring the relative position information of the words in the text.

Suppose we create an instance of CountVectorizer. Then, we can use methods:

- fit(…) to learn the vocabulary from …
- transform(…) to build a document-term matrix from …

Problem 4 [Multivariate Gaussian]

The multivariate Gaussian is defined by:

$$p(\mathrm{x};\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}} e^{-\frac{1}{2}(\mathrm{x}-\mu)^{\top}\Sigma^{-1}(\mathrm{x}-\mu)}$$

Suppose $\mu = \begin{bmatrix} 175 \\ 80 \end{bmatrix}$ is the mean height and weight of some population of people, and the covariance is $\Sigma = \begin{bmatrix} 100 & 0 \\ 0 & 25 \end{bmatrix}$. This diagonal covariance matrix would imply that height and weight are uncorrelated, that height has a standard deviation of 10 cm, and weight has a standard deviation of 5 kg. Execute the following code and see how the matrices effect on the distribution.

```
import numpy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

mu = numpy.matrix([[175], [80]])
Sigma = numpy.matrix([[100, 0], [0, 25]])


def plot_gauss2d(mu, Sigma):
    X = numpy.arange(145, 205.2, 0.2)
    Y = numpy.arange(65, 95.1, 0.1)
    X, Y = numpy.meshgrid(X, Y)
    siginv = numpy.linalg.inv(Sigma)
    xmmu = X - mu[0, 0]
    ymmu = Y - mu[1, 0]
    xts1 = xmmu * siginv[0, 0] + ymmu * siginv[1, 0]
    xts2 = xmmu * siginv[1, 0] + ymmu * siginv[1, 1]
    d = -1 / 2 * (xts1 * xmmu + xts2 * ymmu)
    Z = 1 / (2 * numpy.pi) / numpy.sqrt(numpy.linalg.det(Sigma)) * numpy.exp(d)
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.view_init(60, 280)  # Elevation, Azimuth
    surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                           linewidth=0, antialiased=False)
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.show()


plot_gauss2d(mu, Sigma)
```

You should obtain a plot as follows (see Figure 2):

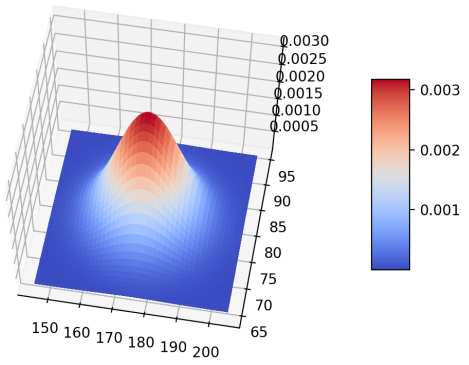

Figure 2 Example of a multivariate Gaussian distribution

It is not realistic to assume that height and weight are uncorrelated. The off diagonal element $\sigma_{12} = \sigma_{21}$ of our 2×2 covariance matrix represents:

$$E[(x_1 - E[x_1])(x_2 - E[x_2])]$$

which when normalized by the variances of the two variables would give us Pearson's correlation between the two variables:

$$r = \frac{\sigma_{12}}{\sqrt{\sigma_{11}\sigma_{22}}}$$

so if, for example, we let $\sigma_{12} = 25$, we would have a correlation of 0.5 between height and weight *i.e.* try changing the value of Sigma as follows:

```
Sigma = numpy.matrix([[100, 25],[25, 25]])
```

Then, we would get a plot as in Figure 3.

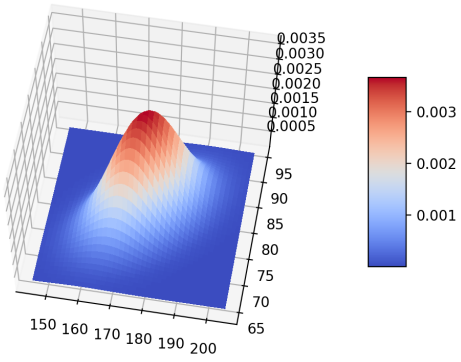

Figure 3 Another example of a multivariate Gaussian distribution

Try some other values for the covariance until you have a good feeling for how the matrix affects the layout of the probability density.

Appendix

Theorem 1. When A is a symmetric matrix and $\alpha = x^T A x$ where x is $n \times 1$, A is $n \times n$, and A does not depend on x, then we have:

$$\frac{\partial \alpha}{\partial x} = 2x^T A$$