Summary Notes
EE425X - Machine Learning: A Signal Processing Perspective

Namrata Vaswani

These notes are work in progress. For the latest version at any time, see the link `https://www.dropbox.com/s/kwle0pf09mp3631/Summary_Notes_2.pdf?dl=0`

Acknowledgement

Much of these notes are based on material presented in this Stanford Machine Learning class `http://cs229.stanford.edu/syllabus.html`.

1 Notation

In these notes $'$ and ^{T} are both used to denote vector or matrix transpose. At a few other places too MATLAB notation is used.

2 Supervised Learning

Given observed data (or features of the observed data) or other "input" $\boldsymbol{x}$, the goal is to predict some function of $\boldsymbol{x}$ that is denoted by y (often called "output"). In what we have talked so far $\boldsymbol{x}$ is an $n \times 1$ vector and y is a scalar. For example, $\boldsymbol{x}$ can be the feature vector of key attributes of a house, while y can be its price. In this case both are real valued. Or y can be a binary decision about whether a buyer buys the house or not.

In learning, we first decide a modeling strategy to model the input-output relationship; then come up with an algorithm to "learn" parameters given the training data (which is a set of m input-output pairs in case of supervised learning). All of this is done so that the "learnt model" can be used to predict y (get $\hat{y}$) for a new query $\boldsymbol{x}$.

"Predict" is often also called "estimate" (if y is real-valued) and it is also called "detect" or "classify" (if y is binary/discrete-valued).

Learning algorithms can be supervised or unsupervised. In supervised learning, we are provided with "training data" that allows us to "learn" the parameters used by the model that our algorithm relies on. Goal is to predict y using observed data or features $\boldsymbol{x}$.

$\boldsymbol{x}$ is $n \times 1$, y is a scalar. We use θ to denote the set of parameters used by our assumed model. The number of parameters (length of θ) can be n or more or less.

In many settings, the assumed model that predicts y is denoted $h_\theta(\boldsymbol{x})$. Since the model is never perfect, we assume that the "true" output y satisfies

$$y = h_\theta(\boldsymbol{x}) + e$$

where e is the *modeling error or noise.* This is typically modeled as a random variable with a probability density function (PDF), typically zero mean Gaussian and independent and identically distributed (i.i.d.) in each new sample.

Training data consists of m input/output pairs $\{\boldsymbol{x}^{(i)}, y^{(i)}\}, i = 1, 2, \ldots, m$. The modeling error / noise e We use these to "learn" θ. Once that is done, we can predict y from $\boldsymbol{x}$ using the above equation.

3 Supervised Learning: Linear Regression

In the setting we have talked about in class, $\boldsymbol{x}$ is a real-valued $n \times 1$ vector and y is a real-valued scalar [1]. The parameter vector θ is also an $n \times 1$ vector

3.1 Model

In linear regression, $h_\theta(\boldsymbol{x})$ is a linear function of $\boldsymbol{x}$.

$$h_\theta(\boldsymbol{x}) = \theta^T \boldsymbol{x}.$$

(I sometimes may use $'$ for transpose – MATLAB notation)

3.2 Learning θ: minimize squared loss

The most common approach to learn θ is to assume a squared error loss and try to minimize it, i.e., find

$$\arg\min_\theta J(\theta) := \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \theta' \boldsymbol{x}^{(i)})^2.$$

Define an $m \times n$ input data matrix $\boldsymbol{X}$ with $(\boldsymbol{x}^{(i)})^T$ as its rows, i.e., let

$$\boldsymbol{X} = \begin{bmatrix} -(\boldsymbol{x}^{(1)})^T - \\ -(\boldsymbol{x}^{(2)})^T - \\ \vdots \\ -(\boldsymbol{x}^{(m)})^T - \end{bmatrix} \tag{1}$$

and define an $m \times 1$ vector $\boldsymbol{y}$ with $y^{(i)}$ as its columns.

Then, $J(\theta)$ can be expressed more compactly as

$$J(\theta) := \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \theta^T \boldsymbol{x}^{(i)})^2 = \frac{1}{m}\|\boldsymbol{y} - \boldsymbol{X}\theta\|_2^2$$

3.2.1 Understanding squared loss: Maximum Likelihood Estimation under i.i.d. Gaussian model

The above can be motivated as Maximum Likelihood Estimation: maximize $p(\boldsymbol{y}; \boldsymbol{X}, \theta)$ under the model $y^{(i)} = h_\theta(\boldsymbol{x}^{(i)}) + e^{(i)},\ i = 1, 2, \ldots, m$ with $w^{(i)}$ independent identically distributed (i.i.d.) standard Gaussian. Here the randomness is only in the noise e.

3.3 Solutions for minimizing squared loss: also called Least Squares (LS) Estimation

1. We can get a closed form solution by taking the derivative of $J(\theta)$ w.r.t. θ and setting it to zero. When $\boldsymbol{X}$ is full rank n (a necessary condition for this is $m \geq n$), this simplifies to

$$\hat{\theta} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}$$

2. For an $m \times n$ matrix with $m \geq n$, $(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'$ is the pseudo-inverse of $\boldsymbol{X}$, denoted $\boldsymbol{X}^\dagger$. Thus, we can also write the above solution as

$$\hat{\theta} = \boldsymbol{X}^\dagger \boldsymbol{y}$$

[1] In more general settings y can also be a real-valued vector (this will not be discussed in our class).

Extra details: The pseudo-inverse is computed by first computing the singular value decomposition (SVD) of the matrix $\boldsymbol{X}$. Suppose the SVD of $\boldsymbol{X} = U\Sigma V'$ where U is an $m \times m$ unitary matrix. V is an $n \times n$ unitary matrix and Σ is diagonal with non-negative entries. Then $\boldsymbol{X}^\dagger = V\Sigma^\dagger U^T$. For a rectangular diagonal matrix such as Σ, we get the pseudo-inverse by taking the reciprocal of each non-zero element on the diagonal, leaving the zeros in place, and then transposing the matrix.

If algorithms for computing matrix inverse or matrix pseudo-inverse were "exact" (and not iterative) then the above two approaches would return the exact same solution. But they are not. Thus, when $\boldsymbol{X}$ is "well-conditioned", both approaches above return same solution, but not otherwise.

3. For large sized problems (where n is large), using Gradient Descent (GD) is a better idea. Since problem is convex, GD should, in principle, converge to above solution starting from any initialization, and should converge pretty quickly. We explain this in Sec. 5.

4. *Approximate but even faster solution:* Stochastic GD (S-GD) can be used. Advantages: faster per iteration; needs lesser memory; and is useful to get a fully streaming algorithm. But no easy guarantees on whether it will converge and to what.

3.4 Including a nonzero mean in the model

To include a non-zero mean in the model, one can replace both $\boldsymbol{x}$ and θ by $n+1$ length vectors as follows. Let

$$\tilde{\theta} = \begin{bmatrix} \theta_0 \\ \theta \end{bmatrix}$$

and

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$$

and we let

$$h_{\tilde{\theta}}(\tilde{\boldsymbol{x}}) = \tilde{\theta}^T \tilde{\boldsymbol{x}}.$$

With this model, we do everything explained above using $\tilde{\boldsymbol{x}}$ and $\tilde{\theta}$ to replace θ and $\boldsymbol{x}$ respectively.

In Homework 1, the above is NEEDED for the part where we generate data using a nonzero mean μ_x.

4 Simulating data: a good code-writing practice

Consider the house price prediction based on house features example. In the previous section, we said we use linear regression to model the data and predict the price. If I write code to learn theta and apply it to a real dataset directly and suppose it does not "work too well" (my code does not give very good predictions on test data). How do I know if (i) the linear regression model is wrong or (ii) my learning approach (normal equations or GD) is wrong, or (iii) there is a code bug (I have an extra factor of 2 at some place by mistake)?

A partial fix to the above problem to above it is to first simulate data using the linear regression model. So when I test my learning code on this data I know what is true θ I am looking for. Then I can try to fix (ii) and (iii) issues. Once these are fixed, then we try the same code on real data and then maybe compare with another model to check which one is better.

4.1 Understanding a multivariate Gaussian distribution

`http://cs229.stanford.edu/section/gaussians.pdf`
`http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-prob.pdf`

4.2 How to simulate

Task: Generate your own data to simulate the linear regression model $y = \boldsymbol{\theta}^T \boldsymbol{x} + e$. Generate m such independent training data vectors. Also generate m_{test} independent data vectors for the testing step.

Let $\mathcal{N}(\mu, \boldsymbol{\Sigma})$ denote the Gaussian distribution with mean μ and covariance $\boldsymbol{\Sigma}$.

Do this as follows:

(1) Fix $\boldsymbol{\theta}$ once as specified below.

(2) Training data generate: For $i = 1$ to m, generate $\boldsymbol{x}^{(i)} \sim \mathcal{N}(0, I)$, $e^{(i)} \sim \mathcal{N}(0, \sigma_e^2)$ and $y^{(i)} = \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} + e^{(i)}$.

(3) Test data generate: In a different Loop, repeat (b) for $i = 1$ to m_{test}. Use this data in the testing step and not for training.

Generate the data for the following settings:

(a) Use $m = 30, 100, 1000$, $n = 5$, and set $e = 0$. Let $\boldsymbol{\theta} = [1, 4, 2, 10, 23]^T$.

(b) Pick $m = 30, 100, 1000$, $n = 5$, and set $\sigma_e^2 = 10^{-6}$. Let $\boldsymbol{\theta} = [1, 4, 2, 10, 23]^T$.

(c) Pick $m = 100, 100, 1000$, $n = 5$, and set $\sigma_e^2 = 10^{-4}$. Let $\boldsymbol{\theta} = [1, 4, 2, 10, 23]^T$.

4.2.1 Notes

For generating the data, I have suggested using the Gaussian distribution just as an example. But you do not have to use the Gaussian. You could also use any other distribution, e.g., the uniform distribution.

On the other hand, for the error $e^{(i)}$, we have assumed that it is Gaussian in our model (that is why the squared loss is justified). So for generating error, you *do have to* use the Gaussian distribution.

5 Gradient Descent (GD) and Stochastic GD

The GD approach is an iterative algorithm to find a local minimizer of a cost function. Which minimizer is found depends on how one initializes. It does not always converge and of course local minimizer may not be a global minimizer either. But if cost function is convex, it will converge to a global minimizer. Moreover if minimizer is also unique (cost is strictly convex), then the only correct solution can be found. An example of this is the squared loss for linear regression. It is convex always. It is strictly convex if $m \geq n$ and $\boldsymbol{X}$ has full rank n.

The GD Algorithm to minimize any cost $J(\theta)$ is as follows. Recall

$$J(\theta) := \frac{1}{m} \sum_{i=1}^{m} cost(\theta, \boldsymbol{x}^{(i)}, y^{(i)})$$

where *cost* is squared error (as in previous section) or can be something else (as in later sections on Logistic Regression).

1. Initialize $\hat{\theta}_0$ as the zero vector (or anything else).
2. Repeat the following for iterations $t > 0$ until "stopping criterion" is reached:

$$\hat{\theta}_t = \hat{\theta}_{t-1} - \mu \nabla_\theta J(\hat{\theta}_{t-1})$$

increment t by one and repeat until stopping criterion reached:

A typical stopping criterion: stop when $\|\hat{\theta}_t - \hat{\theta}_{t-1}\|_2 / \|\hat{\theta}_{t-1}\|_2 \leq \epsilon$ with ϵ a very small tolerance, e.g., set $\epsilon = 10^{-8}$.

Setting μ : when using $J(\theta)$ as above (it is average of the cost functions), then its gradient also contains a $(1/m)$ term, then μ can be a small constant between zero and one, e.g. try $\mu = 0.1$ and then reduce or increase depending on what happens.

Reduce it if the cost seems to not decrease at all or starts increasing. Increase it if the cost decreases but very slowly with iteration.

5.1 Stochastic and Mini-Batch GD

$J(\theta)$ is typically an average of m terms; in fact it always is under our assumption of different training data points being i.i.d. As a result, its gradient is also a sum of m terms divided by m.

If m is large, computing the full gradient at each iteration can be expensive. Also, sometimes not all data is available immediately.

Stochastic GD idea: sum over a subset of the m gradients at each iteration. Pick this subset randomly or use other strategies. See videos on Stochastic GD and Mini-Batch GD.

6 Supervised Learning: Logistic Regression

In this case, $\boldsymbol{x}$ is still a real-valued $n \times 1$ vector but now y is a binary scalar.

6.1 Model

This assumes that $\Pr(y = 1; x, \theta) = h_\theta(\boldsymbol{x})$ with

$$h_\theta(\boldsymbol{x}) = g(\theta^T \boldsymbol{x}), \;\; g(z) := \frac{1}{1 + e^{-z}}$$

$g(.)$ is called the sigmoid function, it takes values between zero and one for all values of z. Thus, it can be used to model a probability. Said another way,

$$p(y; \boldsymbol{x}, \theta) = h_\theta(\boldsymbol{x})^y (1 - h_\theta(\boldsymbol{x}))^{1-y}$$

The prediction is

$$\hat{y} = \arg\max_{y=0,1} p(y; \boldsymbol{x}, \theta)$$

Thus

$$\hat{y} = 1 \text{ if } h_\theta(\boldsymbol{x}) > 1 - h_\theta(\boldsymbol{x}),$$

and $\hat{y} = 0$ otherwise.

6.2 Use of bias term in Logistic regression

Introduce the bias term exactly as we did in case of linear regression Make both θ and $\boldsymbol{x}$ $(n+1)$-th length vectors; set the first entry of $\boldsymbol{x}$ equal to 1. The first entry of θ is then the bias term.

6.3 Learning θ: Maximum Likelihood Estimation

Again define $\boldsymbol{y}$ and $\boldsymbol{X}$ as before from training data

Use Maximum Likelihood Estimation again: assume i.i.d. training data points $y^{(i)}$ (recall that this was assumed also in linear regression – it was imposed by letting the $w^{(i)}$'s be i.i.d.).

Thus, we minimize the negative log likelihood,

$$J(\theta) := -\log p(\boldsymbol{y}|\boldsymbol{X}; \theta) = -\log \left(\prod_{i=1}^{m} p(\boldsymbol{y}_i; \boldsymbol{x}^{(i)}, \theta) \right) = -\log \left(\prod_{i=1}^{m} h_\theta(\boldsymbol{x}^{(i)})^{y^{(i)}} (1 - h_\theta(\boldsymbol{x}^{(i)}))^{1-y^{(i)}} \right)$$

We can simplify this a lot as follows.

$$\begin{aligned} J(\theta) &= -\sum_{i=1}^{m} \log \left(\left(\frac{1}{1 + \exp(\theta' \boldsymbol{x}^{(i)})} \right)^{y^{(i)}} \left(\frac{\exp(\theta' \boldsymbol{x}^{(i)})}{1 + \exp(\theta' \boldsymbol{x}^{(i)})} \right)^{1-y^{(i)}} \right) \\ &= -\sum_{i=1}^{m} \log \left(\left(\frac{1}{1 + \exp(\theta' \boldsymbol{x}^{(i)})} \right)^{y^{(i)}} \left(\frac{1}{1 + \exp(-\theta' \boldsymbol{x}^{(i)})} \right)^{1-y^{(i)}} \right) \end{aligned}$$

The only difference between the first and second terms inside log(.) is the sign of $\theta' \boldsymbol{x}^{(i)}$ in the denominator and the power it is raised to. One is raised to the power $y^{(i)}$, the other is raised to the power $(1-y^{(i)})$. Here $y^{(i)}$ is either 0 or 1. So, $2y^{(i)}-1$ is either -1 (when $y^{(i)}=0$) or $+1$ (when $y^{(i)}=1$). Thus, when $y^{(i)}$ takes only values 0 or 1,

$$\left(\frac{1}{1+\exp(\theta'\boldsymbol{x}^{(i)})}\right)^{y^{(i)}}\left(\frac{1}{1+\exp(-\theta'\boldsymbol{x}^{(i)})}\right)^{1-y^{(i)}}=\frac{1}{1+\exp\left((\theta'\boldsymbol{x}^{(i)})(2y^{(i)}-1)\right)}$$

The reason the last equality is true is because when $y^{(i)}=1$, $2y^{(i)}-1=1$, but when when $y^{(i)}=0$, $2y^{(i)}-1=-1$.

We now simply get

$$J(\theta)=-\sum_{i=1}^{m}\log\left(\frac{1}{1+\exp\left((\theta'\boldsymbol{x}^{(i)})(2y^{(i)}-1)\right)}\right)=\sum_{i=1}^{m}\log\left(1+\exp\left((\theta'\boldsymbol{x}^{(i)})(2y^{(i)}-1)\right)\right)$$

We can find θ by minimizing $J(\theta)$ by GD.

It is possible to show that $J(\theta)$ is convex. Argument: weighted sum convex functions is convex when the weights are positive, here the weights are just 1; $(\theta'\boldsymbol{x})(2y-1)$ is an affine function of θ (and hence is both convex and concave; the logistic function $\log(1+\exp(-z))$ can be shown be convex (see next line); composition of a convex function and an affine function is convex. To show $\log(1+\exp(-z))$ is convex, since it is twice differentiable, we can compute the second derivative and show that it is $-1/(1+\exp(-z))^2<0$ for any z, thus it is convex everywhere.

7 Supervised Learning: Generative Learning (a.k.a. Bayesian models/learning)

For logistic or linear regression we just assumed a probabilistic model on how y is generated from $\boldsymbol{x}$, with $\boldsymbol{x}$ being deterministic.

In Generative Learning, we assume a "generative model": we first put a prior probabilistic model on y, and then assume a probabilistic model on how $\boldsymbol{x}$ was generated from y. We then compute the probability (or probability density function in case y is real-valued) of y taking a certain value given $\boldsymbol{x}$ using Bayes rule. Mathematically, we assume that we are given

$$p(\boldsymbol{x}|y;\theta), p(y)$$

and we use these to obtain the prediction as follows

$$\hat{y}=\arg\max_y p(y|\boldsymbol{x};\theta):=\arg\max_y \frac{p(\boldsymbol{x}|y;\theta)p(y;\theta)}{p(\boldsymbol{x};\theta)}=\arg\max_y p(\boldsymbol{x}|y;\theta)p(y;\theta)$$

This use of Bayes rule is called **Maximum A Posteriori (MAP)** detection or estimation in other literature. The overall approach is often called Bayesian modeling or physics-based modeling.

7.1 Learning θ: Maximum Likelihood Estimation

Estimate θ: define $\boldsymbol{y}$, $\boldsymbol{X}$ as before from training data. Also assume training data points are independent: $\{\boldsymbol{x}^{(i)}, y^{(i)}\}$ are mutually independent for different i. Define the cost function

$$J(\theta):=\Pr(\boldsymbol{y},\boldsymbol{X};\theta)=\prod_{i=1}^{m}p(y^{(i)},\boldsymbol{x}^{(i)};\theta)$$

or, usually its logarithm, and maximize it over θ.

7.2 Generative Learning: Gaussian Discriminant Analysis (GDA)

This is one type of generative model and learning algorithm for the setting $\boldsymbol{x}$ real-valued $n \times 1$ vector and y binary scalar. Thus y can take two values 0 or 1. It assumes $\boldsymbol{x}$ is Gaussian given y and y itself is Bernoulli, i.e.,

$$p(\boldsymbol{x}|y;\theta) = \mathcal{N}(\boldsymbol{x};\mu_y,\boldsymbol{\Sigma}_y),\ p(y) = \phi^y(1-\phi)^{1-y}$$

Notice in this case $\theta = \{\mu_0, \mu_1, \boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1, \phi\}$. Parameters are still learnt by MLE

$$\max_\theta J(\theta) := \Pr(\boldsymbol{y}, \boldsymbol{X};\theta) = \prod_{i=1}^m p(y^{(i)}, \boldsymbol{x}^{(i)};\theta) \text{ s.t. } 0 \le \phi \le 1$$

Notice that, without extra assumptions, we have $2n + 2n^2 + 1$ parameters. Training data are each n length vectors $\boldsymbol{x}^{(i)}$, thus we can say we have mn training data scalars. We need mn significantly larger than $2n + 2n^2 + 1$ for training/learning to be accurate. We will need m growing at least linearly with n to be able to learn anything useful.

But the point of Bayesian (generative) modeling is that we should be able to use a smaller m and still train well.

When enough training data is not available, we need to simplify our model so that there are fewer parameters. As explained later, this will increase model bias, but will reduce the variance in parameter estimation.

A common model simplification is to assume that the different entries of each $\boldsymbol{x}^{(i)}$ are independent conditioned on the class label $y^{(i)}$. This is called the *Naive Bayes assumption*.

7.2.1 Gaussian Discriminant Analysis (GDA) with Naive Bayes assumption and equal covariances

A common model simplification is to assume that the different entries of each $\boldsymbol{x}^{(i)}$ are independent conditioned on the class label $y^{(i)}$. This is called the *Naive Bayes assumption*.

In the Gaussian case, this translates to assuming that Σ_0, Σ_1 are *diagonal*. With the diagonal assumption, we now have only $2n + 2n + 1$ parameters which is much more manageable. A second commonly used simplification is to assume the same covariance under both classes, i.e., that $\Sigma_0 = \Sigma_1 = \Sigma$ and Σ is diagonal. With this assumption too, we have the following simpler model

$$\prod_{i=1}^m \left(\left(\prod_{j=1}^n \mathcal{N}(\boldsymbol{x}_j; (\mu_{y_i})_j, \sigma_j^2) \right) \cdot \phi^{y_i}(1-\phi)^{1-y_i} \right)$$

Under the above assumption, the Max Likelihood Estimates (MLE) of the model parameters, $\theta := \{\phi, \mu_0, \mu_1, \Sigma\}$ i.e., the value of the model parameters that solve

$$\arg\max_\theta J(\theta),\ J(\theta) := \Pr(\boldsymbol{y}, \boldsymbol{X};\theta) = \prod_{i=1}^m p(y^{(i)}, \boldsymbol{x}^{(i)};\theta) \text{ s.t. } 0 \le \phi \le 1$$

are computed as follows:

$$\hat{\phi} = \frac{1}{m}\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)$$

$$\hat{\mu}_0 = \frac{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 0)\boldsymbol{x}^{(i)}}{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 0)}$$

$$\hat{\mu}_1 = \frac{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)\boldsymbol{x}^{(i)}}{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)}$$

$$\hat{\sigma_j^2} = \frac{1}{m}\sum_{i=1}^m (\boldsymbol{x}^{(i)} - \mu_{y^{(i)}})_j^2,\ j = 1, 2, \ldots, n$$

while setting all non-diagonal entries of $\hat{\Sigma}$ to be zero. Here $\mathbf{1}$ denotes the *indicator function* of the statement in paranthesis. Thus, $\mathbf{1}(y^{(i)} = 0)$ equals one if $y^{(i)} = 0$ and it equals zero otherwise.

Notice that the above is equivalent to learning the parameters for *each feature* independently, it can also be rewritten as follows: for each $j = 1, 2, \ldots, n$, compute

$$(\hat{\mu}_0)_j = \frac{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 0)(\boldsymbol{x}^{(i)})_j}{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 0)}$$
$$(\hat{\mu}_1)_j = \frac{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)(\boldsymbol{x}^{(i)})_j}{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)}$$
$$\hat{\sigma_j^2} = \frac{1}{m}\sum_{i=1}^m (\boldsymbol{x}^{(i)} - \mu_{y^{(i)}})_j^2, \ j = 1, 2, \ldots, n$$

In this Naive Bayes' setting, the selecting of $\hat{y}$ to solve

$$\hat{y} = \arg\max_y p(\boldsymbol{x}|y; \hat{\theta})p(y; \hat{\theta})$$

simplifies to the following test (just simplify the expressions):

$$\hat{y} = 1 \text{ if } \sum_{j=1}^n \frac{(\boldsymbol{x} - \hat{\mu}_1)_j^2}{\hat{\sigma}_j^2} < \sum_j \frac{(\boldsymbol{x} - \hat{\mu}_0)_j^2}{\hat{\sigma}_j^2}$$

7.2.2 Notes

1. In the Gaussian case, we get a simple closed form as above. In many cases of Bayesian modeling also, this is possible. In general when this is not possible, *do not* ever work directly with probabilities. *Always work with logarithms of the probabilities.* Otherwise you will run into numerical problems while coding. Remember under the naive Bayes assumption, $p(\boldsymbol{x}|y; \hat{\theta}) = \prod_{j=1}^n p(\boldsymbol{x}_j|y; \hat{\theta})$ if $p(.)$ is a PMF, it is a product of n real numbers all less than one. Even if $p(.)$ is a PDF (which could be more than one), very often it is actually less than one only. Product n numbers less than one can become very small very soon.

2. When working with real data, e.g., images, there may be a certain region that is black in *all* the images. For these pixels the estimate of the variance will be zero meaning $\sigma_j^{-1} = \infty$ resulting in code bugs.

Fix 1: do not use these directions.

Fix 2: Better fix to deal with similar issues where in some directions variance is very small (may not be zero): use PCA to reduce the dimensionality of the data. For classification there is no need to use all n features.

Fix 3: add a small value to replace the zeros: the added value should be *much smaller* than any of the *important directions' variances.* This is easy in the zero/nonzero case but in practice ill-conditioning of Σ may make this hard to do. Use of PCA and a smaller dimension is thus a much better fix.

7.3 Generative Learning: Spam Filter – an example of Discrete-valued or Categorical Features

In applications such as spam email detector (or filter) design, one typically models $\boldsymbol{x}$ as a discrete-valued vector given y.

$y = 0$ means the email is not-spam, $y = 1$ means it is spam.

7.3.1 Simple Spam Filter: entries of x are binary

In the simplest version, $\boldsymbol{x}$ is an $n \times 1$ binary vector with n being equal to the size of the English dictionary. We say $\boldsymbol{x}_j = 1$ if the j-th dictionary word is in the email and $\boldsymbol{x}_j = 0$ otherwise. This means n is really large. Also it is not counting how many times a word occurred.

Consider first the simplest model where $\boldsymbol{x}$ is a binary vector. As before, we specify

$$p(\boldsymbol{x}|y;\theta),\ p(y)$$

and we predict

$$\hat{y} = \arg\max_y p(\boldsymbol{x}|y;\theta)p(y;\theta).$$

Notice now that $\boldsymbol{x}$ can take a total of 2^n possible values. We also need to specify the prior $p(y)$. Thus, in this most general case, the number of parameters equals $2^n + 1$.

Training data are each n length vectors $\boldsymbol{x}^{(i)}$, thus we can say we have mn training data scalars. We need mn significantly larger than $2^n + 1$ for training/learning to be accurate. Here we will need m to grow linearly with 2^{n-1}: this can be very large and is not practical.

7.3.2 Naive Bayes assumption

In both the above examples and especially the second one, the required m can be very large for accurate training/learning. Thus we add a further modeling assumption called "Naive Bayes" in ML literature. Others would call it *"conditional independence" of different entries of a feature vector (the different $\boldsymbol{x}_j$'s, $j = 1, 2, \ldots, n$) given y.* Mathematically, we are assuming

$$p(\boldsymbol{x}|y;\theta) = \prod_{j=1}^{n} p(\boldsymbol{x}_j|y;\theta)$$

This may not be a very realistic assumption, but it significantly reduces the number of parameters required by the model.

In the Gaussian case, this implies that $\boldsymbol{\Sigma}_0$, $\boldsymbol{\Sigma}_1$ are **diagonal matrices**. Thus, the number of parameters becomes $2n + 2n + 1$ which is much more tractable. In the spam filter case, this means we have $n + 1$ parameters.

So now the number of training samples m does not even need to grow with n.

7.3.3 Simple Spam filter with Naive Bayes

$\boldsymbol{x}$ is a binary vector, y is a scalar. With using Naive Bayes, in the Simple Spam Filter case, we can now define

$$\psi_{j,y} := p(\boldsymbol{x}_j = 1|y), j = 1, 2, \ldots, n;\ \textit{and}\ \phi := p(y = 1)$$

With this we have just $n + 1$ parameters to learn instead of $2^n + 1$.

We can again learn the parameters by MLE:

$$\max_\theta J(\theta) := \prod_{i=1}^{m} p(y^{(i)}, \boldsymbol{x}^{(i)};\theta) = \prod_{i=1}^{m}\prod_{j=1}^{n} p(\boldsymbol{x}_j^{(i)}|y^{(i)};\theta)p(y^{(i)};\theta) = \prod_{i=1}^{m} \phi^{y^{(i)}}(1-\phi)^{(1-y^{(i)})}(\prod_{j=1}^{n} \psi_{j,y}^{\boldsymbol{x}_j^{(i)}}(1-\psi_{j,y})^{(1-\boldsymbol{x}_j^{(i)})}$$

s.t. constraints that

$$0 \le \psi_{j,y} \le 1,\ 0 \le \phi \le 1$$

Can again get closed form simple expressions for the MLE:

$$\hat{\psi}_{j,0} =$$

count the number of training data points for which $y^{(i)} = 0$ and $\boldsymbol{x}_j^{(i)} = 1$ and divide by the total number of training data points for which $y^{(i)} = 0$.

Similarly

$$\hat{\psi}_{j,1} =$$

count the number of training data points for which $y^{(i)} = 1$ and $\boldsymbol{x}^{(i)} = 1$ and divide by the total number of training data points for which $y^{(i)} = 1$. and

$$\hat{\phi} =$$

count the number of training data points for which $y^{(i)} = 1$ and divide by m

Use of n to be dictionary size makes it extremely large causing the algorithm to be very slow. Also, instead of letting n be the size of the English dictionary, we can let n be the size of the vocabulary (set of all words in all training data). But this has the disadvantage that it does not tell you how to deal with an unseen word. Options include (i) ignore unseen words (can be problematic in case the unseen words are the only reason an email is obviously spam); or (ii) in the model, assume a small nonzero probability for an unseen word (this means: increase the vocabulary size by 1, this probability cannot be learned easily, you just have to make up a "reasonable" value for it).

7.3.4 Spam Filter: entries of z count the number of times a word occurs

The simplest spam filter explained above is not using word-counts (the number of times a word occurs), but just checking whether a word is present or not. A better model is to consider a different feature vector $\boldsymbol{z}$ with $\boldsymbol{z}_j$ being the number of times word j occurs in the vocabulary (or dictionary).

Treating each word occurrence as independent (this is an assumption (naive Bayes), it is actually not true since certain pairs of words are much more likely occur together, but simplifies our modeling), and assuming as before that $\psi_{j,y} = \Pr(\text{word } j \text{ is present in the email}|y)$, the feature vector $\boldsymbol{z}$ would be modeled by what is called a "multinomial distribution" as follows. This assumes that there are a total of d words in the sample, i.e., that $\sum_j \boldsymbol{z}_j = d$

$$p(\boldsymbol{z}|y) = \binom{d}{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n} \prod_{j=1}^{n} \psi_{j,y}^{\boldsymbol{z}_j}$$

This is called the multinomial distribution with parameters $\psi_{j,y}$.

MLE The joint likelihood of m independent training samples can be expressed as follows: let $d^{(i)}$ be the number of words in training sample i. Then,

$$J(\theta) = \prod_{i=1}^{n} p(y^{(i)}) p(\boldsymbol{z}^{(i)}|y^{(i)}) = \prod_{i=1}^{n} \phi^{y^{(i)}} (1-\phi)^{(1-y^{(i)})} \binom{d^{(i)}}{\boldsymbol{z}_1^{(i)}, \boldsymbol{z}_2^{(i)}, \ldots, \boldsymbol{z}_n^{(i)}} \prod_{j=1}^{n} \psi_{j,y^{(i)}}^{\boldsymbol{z}_j^{(i)}}$$

If we want all $d^{(i)}$s to be equal to d, we then use the blank-space as one feature.

Learning parameters: we need to maximize the above over $\phi, \psi_{j,y}$ subject to the constraints that

$$0 \leq \psi_{j,y} \leq 1, \ 0 \leq \phi \leq 1$$

It can be shown that we get

$$\hat{\psi}_{j,0} = \frac{\sum_{i=1}^{m} \boldsymbol{z}_j^{(i)} \mathbb{1}(y^{(i)} = 0)}{\sum_{j=1}^{n} \sum_{i=1}^{m} \boldsymbol{z}_j^{(i)} \mathbb{1}(y^{(i)} = 0)}$$

In words this is the total number of times word j occurs in **all** emails that are not spam (for which $y^{(i)} = 0$) divided by the total number of words in **all** emails that are not spam. We can similarly compute

$$\hat{\psi}_{j,1}$$

7.3.5 Spam filter: model 3

A third possible model is as follows. Let j referred to the j-th word in the email and let $\boldsymbol{x}_j$ be the location of the j-th word in the dictionary. So then this becomes "categorical data". We explain below how to deal with categorical data. In this case the length of the feature vector can be different for different emails. It will be $n^{(i)}$ if the email has $n^{(i)}$ distinct words.

7.3.6 More ideas and related problems

A more advanced document model models co-occurrences of words.

Similar ideas are also used to classify blogs or webpages into various categories as well.

7.4 Modeling Categorical data

In some problems, the different features may be "categorical" instead of binary, which means feature j takes one of K_j possible values. For example $\boldsymbol{x}_j$ could be color of the front door of a house in case of the house price example and we assume $K_j = 5$ possible colors for example. There is no ordering to which color is preferred, thus the integer labels are arbitrary; they do not have a numerical meaning.

The model would then require us to learn $\psi_{j,y,k} := \Pr(\boldsymbol{x}_j = k|y)$ for $k = 1, 2, \ldots K_j$, $y = 0, 1$, $j = 1, 2, \ldots, n$.

Our data model then is as follows

$$p(\boldsymbol{x}|y) = \prod_{j=1}^{n} \prod_{k=1}^{K_j} \psi_{j,y,k}^{[\boldsymbol{x}_j == k]}$$

where $[x == k]$ takes the value 1 if $x = k$ and zero otherwise (MATLAB notation). Thus for MLE we need to maximize

$$\prod_i \left(\prod_{j=1}^{n} \prod_{k=1}^{K_j} (\psi_{j,y^{(i)},k})^{[\boldsymbol{x}_j^{(i)} == k]} \phi^{y^{(i)}} (1-\phi)^{1-y^{(i)}} \right)$$

s.t. sum to one constraints on all the probabilities.

Then MLE estimates are given by

$$\hat{\psi}_{j,0,k} =$$

counts the number of times $y^{(i)} = 0$ and $\boldsymbol{x}_j^{(i)} = k$ in the training data and divides this by the number of times $y^{(i)} = 0$ (number of points from class zero in the training data). Do this for every value of $k = 1, 2, \ldots, K_j$ and for every feature $\boldsymbol{x}_j$, $j = 1, 2, \ldots, n$.

Do the same for class label 1.

7.5 Dealing with discrete-valued and real-valued data together in the Generative Learning (a.ka. Bayesian) framework

Often some of the features can be categorical and some of them can be real-valued. Once we impose the Naive Bayes assumption (conditioned on the class label, different features are independent), this is easy to deal with. For simplicity suppose that the first r features are real-valued and the rest $n - r$ are categorical with k_j categories. Also assume the real-valued feature follow a Gaussian distribution with mean $\mu 0$ or μ_1 (depending on the class label) and covariance matrix Σ. By Naive Bayes, Σ is diagonal.

Then, joint likelihood becomes

$$\prod_{i=1}^{m} \left(\left(\prod_{j=1}^{r} \mathcal{N}(\boldsymbol{x}_j^{(i)}; (\mu_{y^{(i)}})_j, \sigma_j^2) \cdot \prod_{j=r+1}^{n} \prod_{k=1}^{K_j} \psi_{j,y^{(i)},k}^{[\boldsymbol{x}_j^{(i)} == k]} \right) \cdot \phi^{y^{(i)}} (1-\phi)^{1-y^{(i)}} \right)$$

s.t. sum to one constraints on ϕ and on all the $\psi_{j,y,k}$'s.

The MLE estimates are computed as follows.

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}(y^{(i)} = 1)$$

For $j = 1, 2, \ldots, r$,

$$(\hat{\mu}_0)_j = \frac{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 0)(\boldsymbol{x}^{(i)})_j}{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 0)}$$
$$(\hat{\mu}_1)_j = \frac{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)(\boldsymbol{x}^{(i)})_j}{\sum_{i=1}^m \mathbf{1}(y^{(i)} = 1)}$$
$$\hat{\sigma_j^2} = \frac{1}{m}\sum_{i=1}^m (\boldsymbol{x}^{(i)} - \mu_{y^{(i)}})_j^2, \ j = 1, 2, \ldots, n$$

For $j = r+1, r+2, \ldots, n$,

$$\hat{\psi}_{j,0,k} =$$

counts the number of times $y^{(i)} = 0$ and $\boldsymbol{x}_j^{(i)} = k$ in the training data and divides this by the number of times $y^{(i)} = 0$ (number of points from class zero in the training data). Do this for every value of $k = 1, 2, \ldots, K_j$ and for every feature $\boldsymbol{x}_j$, $j = 1, 2, \ldots, n$.

8 Reliability of an output

Linear regression predicts a real-valued scalar where as everything we learnt after that predicts a class label (solves a classification problem). Given a query, we can always obtain a prediction. But the other important question to answer is : how reliable is the prediction we obtained. The answer to this question depends on

- the problem itself, e.g., in case of classification by GDA, if the two class means are very close, it is not easy to distinguish the classes. More precisely what matters is how close the class means along a given direction compared to the standard deviation along that direction. Practically this means the following: it easier to distinguish dog pictures from human pictures than from cat pictures

  Similarly for a regression problem, the amount of modeling error e or its variance decides how good the prediction is.

- the number of training data points, and how well the training and test data match (this decides quality of learnt model). In most of what we learn, it is assumed that training and test data are genareted from the same distribution, but in real life this may not be true.

- the specific query: if the query image is of a fluffy cat that may look dog-like, then it is hard to reliably provide a correct classification.

- the last problem can be partly addressed by changing the learning algorithm (the assumed model on the data).

9 Supervised Learning: Linear Classifiers and Support Vector Machines (SVMs)

9.1 Linear Classifiers

Both logistic regression and GDA with Naive Bayes and equal covariances result in a linear classifier, i.e., one can simplify the classification rule in both cases to get the following:

$$\hat{y} = 1 \ \text{ if } \boldsymbol{w}^T\boldsymbol{x} + b > 0$$

and equals 0 otherwise.

Proof for GDA:

GDA decision rule is: $\hat{y} = 1$ if

$$(\boldsymbol{x} - \mu_1)^T \Sigma_1^{-1} (\boldsymbol{x} - \mu_1) < (\boldsymbol{x} - \mu_0)^T \Sigma_0^{-1} (\boldsymbol{x} - \mu_0)$$

With $\Sigma_1 = \Sigma_0 = \Sigma$ and naive Bayes (Σ is diagonal), this simplifies to checking if

$$\sum_j \frac{(\boldsymbol{x} - \mu_1)_j^2}{\sigma_j^2} < \sum_j \frac{(\boldsymbol{x} - \mu_0)_j^2}{\sigma_j^2}$$

Simplifying the above, we equivalently need

$$\boldsymbol{w}^T \boldsymbol{x} + b > 0, \quad \text{with} \quad \boldsymbol{w}_j = \frac{(\mu_1 - \mu_0)_j^2}{\sigma_j^2}, b = 0.5 \sum_j \frac{((\mu_1 - \mu_0)_j)(\mu_1 + \mu_0)_j}{\sigma_j^2}$$

In fact we can also get an expression for $\boldsymbol{w}$ and b even if we just have $\Sigma_1 = \Sigma_0$. We will get $\boldsymbol{w} = \Sigma^{-1}(\mu_1 - \mu_0)$ and $b = \mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0$. ?? check.

9.2 Support Vector Machines (SVMs): Motivation and main idea

Both the classifiers we talked about so far are linear classifiers as explained above. Consider logistic regression. The classification decision would be much more reliable if $\theta^T \boldsymbol{x}$ were either much larger or much smaller than zero.

The idea of SVM is this: we do not assume any data model here. Instead we try to look for the "separating hyperplane", equivalently, a vector θ so that the "margin" from the decision boundary is maximized for *all* training data points. Visual explanation in class or see cs229-notes-3.

9.2.1 Notation change

Instead of a single vector θ with the first entry used for the bias term, in case of SVMs, we use a weight vector $\boldsymbol{w}$ which is the same length as the data and a scalar b. Also, instead of labeling the two classes as 0 and 1, we label them is -1 and $+1$ because this simplifies some of the writing.

Margin: the distance of a data point from the separating hyperplane. Margin for the i-th training data point is computed as

$$y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b)$$

9.2.2 Using SVM for classification

Suppose first that the optimal choice of $\boldsymbol{w}, b$ is available. Then we classify as follows

$$\hat{y} = sign(\boldsymbol{w}^T \boldsymbol{x} + b)$$

If the term ¿ 0, then the class is $+1$ else the class is -1.

9.2.3 Goal

The goal in case of SVMs is to find $\boldsymbol{w}, \boldsymbol{b}$ that maximize the worst-case margin defined by

$$\min_{i=1,2,\ldots,m} y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b)$$

By multiplying $\boldsymbol{w}, b$ by a scalar we could keep increasing the margin, but that will not improve classification. Thus, we need to impose the constraint that $||\boldsymbol{w}||_2 = 1$.

9.3 Simplifications to obtain a convex optimization problem

Writing a slightly different way, we need to solve

$$\max_{\boldsymbol{w},b} \gamma \text{ s.t. } \quad y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq \gamma,\ i = 1, 2, \ldots, m \text{ and } ||\boldsymbol{w}||_2 = 1$$

This is not a convex optimization problem yet. So we try to simplify further. The above is equivalent to dividing everything by $||\boldsymbol{w}||_2$. Doing this gives

$$\max_{\boldsymbol{w},b} \frac{\gamma}{||\boldsymbol{w}||_2} \text{ subject to } y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq \gamma,\ i = 1, 2, \ldots, m$$

This is still not convex. Another point to notice is this: we could pick $||\boldsymbol{w}||_2$ to be anything, so equivalently, we could fix the value of the margin γ to 1, and nothing will change. This gives

$$\max_{\boldsymbol{w},b} \frac{1}{||\boldsymbol{w}||_2} \text{ subject to } y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1,\ i = 1, 2, \ldots, m$$

This is not convex either but now a simple reformulation gives us a convex problem. Maximizing $1/||\boldsymbol{w}||_2$ is equivalent to minimizing $||\boldsymbol{w}||_2$ which is the same as minimizing $||\boldsymbol{w}||_2^2$. This gives

$$\min_{\boldsymbol{w},b} ||\boldsymbol{w}||_2^2 \text{ subject to } y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1,\ i = 1, 2, \ldots, m$$

This is now a convex optimization problem since the cost is a convex function and the inequality constraints are linear. In particular it is what is called a Quadratic Program or QP.

9.4 Final primal problem

$$\min_{\boldsymbol{w},b} ||\boldsymbol{w}||_2^2 \text{ subject to } y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1,\ i = 1, 2, \ldots, m$$

This is now a convex optimization problem since the cost is a convex function and the inequality constraints are linear. In particular it is what is called a Quadratic Program or QP.

9.5 Simplification using duality: needed to develop Kernel SVMs

By using *Lagrange duality*, it is possible to show that we can also compute the optimal $\boldsymbol{w}, b$ as follows.

1. Solve the following "dual problem": optimize over the Langrange multipliers α_i

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - 0.5 \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, \boldsymbol{x}^{(j)} \rangle \right) \text{ s.t. } \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,\ \alpha_i \geq 0,\ i = 1, 2, \ldots, m$$

   (here $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle = \boldsymbol{x}_1^T \boldsymbol{x}_2$)

2. Obtain

$$\hat{\boldsymbol{w}} = \sum_i \hat{\alpha}_i y^{(i)} \boldsymbol{x}^{(i)}$$

$$\hat{b} = -0.5 \left(\max_{i:y^{(i)}=-1} \hat{\boldsymbol{w}}^T \boldsymbol{x}^{(i)} + \min_{i:y^{(i)}=1} \hat{\boldsymbol{w}}^T \boldsymbol{x}^{(i)} \right)$$

   Notice the dependence on feature vectors is only through inner products

Classification:

1. Classification also only uses inner products

$$\hat{\boldsymbol{w}}^T \boldsymbol{x} + \hat{b} = \sum_i \alpha_i y^{(i)} \langle \boldsymbol{x}^{(i)}, \boldsymbol{x} \rangle + \hat{b}$$

The above is useful in two settings: (i) if $n \gg m$, i.e. the original data lies in a much higher dimensional space compared to the available number of data points (this often hapens in classification problems), then the above dual is much less expensive to solve.

(ii) Notice that in the above problem, all dependence on the feature vectors $\boldsymbol{x}^{(i)}$ is through the inner product $\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle$. This means we could consider higher dimensional feature vectors, i.e., convert $\boldsymbol{x}$ to $\phi(\boldsymbol{x})$ by considering features of the form $\boldsymbol{x}_j^2$, $\boldsymbol{x}_j \boldsymbol{x}_k$.

For many such high dimensional feature mappings, $\phi(\boldsymbol{x}^{(i)})$ is very expensive to compute for all training data $i = 1, 2, \ldots, m$. But just computing the inner product, defined as the "kernel product"

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) := \langle \phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2) \rangle$$

is much less expensive. An example is $\phi(x)$ obtainedas all pairwise products of entries of $\boldsymbol{x}$. For this, computing $\phi(\boldsymbol{x}^{(i)})$ takes order n^2 time, thus computing the new feature vector for all i takes order $n^2 m$ time. But computing one kernel product can be done as

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1^T \boldsymbol{x}_2)^2$$

This only takes order n time. There are $m(m+1)/2$ total products to compute. Thus, the time needed is of order nm^2. Since typically, $m \ll n$, this is much quicker.

(iii) It is also possible to define kernel products for cases where the actual feature mapping is infinite dimensional. Gaussian kernel is an example.

9.6 Kernel SVM

A very large number of kernels can be defined. The purpose is for datasets which are not linearly separable in the original feature space, it is possible they are in a higher dimensional space.

Kernel product: is basically some measure of similarity between two data points. Any useful measure of similarity can be used to define a "kernel" (kernel product), one may not even need to specify the underlying feature mapping.

This "kernel trick" can be used for many other learning algorithms as well. Anytme all computation depends on inner products between the features, this can be used.

9.7 Soft margin SVMs

See page 19 of ML-cs229-noted-3

10 Deviation: Introduction to Langrange duality to undertand how to derive the dual program

To be updated later (to make this course 425/525).

See ML-cs229-notes3

11 Supervised Learning: Decision Trees

See handout

12 Cross Validation: leave-one-out-cross-validation

When limited training data is available, one cannot use distinct subsets of the data for training and testing. Use of distinct subsets is easiest to code in but wastes a lot of data. One approach to address this is called Cross-Validation.

The simplest type of cross-validation is leave-one-out-cross validation. I explain it next

alternative goal could be to dimension reduce and find

12.1 Leave-one-out cross validation: any general task

- for $i = 1$ to m
  - at iteration i, define the training data matrix $\boldsymbol{X}$ and vector $\boldsymbol{y}$ as follows: use *all* data except the i-th

$$\boldsymbol{X} = \begin{bmatrix} -(\boldsymbol{x}^{(1)})^T- \\ -(\boldsymbol{x}^{(2)})^T- \\ \vdots \\ -(\boldsymbol{x}^{(i-1)})^T- \\ -(\boldsymbol{x}^{(i+1)})^T- \\ \vdots \\ -(\boldsymbol{x}^{(m)})^T- \end{bmatrix}$$

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}^{(1)} \\ \boldsymbol{y}^{(2)} \\ \vdots \\ \boldsymbol{y}^{(i-1)} \\ \boldsymbol{y}^{(i+1)} \\ \vdots \\ \boldsymbol{y}^{(m)} \end{bmatrix}$$

  - Learn the model parameters use $\boldsymbol{X}, \boldsymbol{y}$ as the training data
  - Use the i-th data points for computing the error $err(i)$.

  end for

Compute

$$TestMSE = \sum_{i=1}^{m} err(i), NormalizedTestMSE = \text{as needed for the problem}$$

12.2 Leave-one-out cross validation for Linear regression

Given m training data points $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{i}i$, $i = 1, 2, \ldots, m$. Suppose the goal is to evaluate the validity of a linear regression model on this data. We will compute Normalized-Test-MSE as follows.

- for $i = 1$ to m
  - at iteration i, define the training data matrix $\boldsymbol{X}$ and vector $\boldsymbol{y}$ as follows: use *all* data except the i-th. This is defined above.
  - Compute $\hat{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
  - Compute $err(i) = (\boldsymbol{y}^{(i)} - \hat{\theta}^T\boldsymbol{x}^{(i)})^2$

  end for

Compute

$$TestMSE = \sum_{i=1}^{m} err(i),\ NormalizedTestMSE = TestMSE/(\sum_{i=1}^{m}(\boldsymbol{y}^{(i)})^2)$$

13 Unsupervised Learning: PCA

In unsupervised learning, there is no *labeled* training data to learn parameters from. So no "output" $y^{(i)}$ is available for "input" $\boldsymbol{x}^{(i)}$. PCA is an unsupervised learning technique that is used for dimension reduction. Given data vectors in $\Re^n$, if they approximately lie in a lower dimensional subspace, how do we find that subspace?

13.1 Why PCA

As before we assume that we are given m data vectors (usually called feature vectors) $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$ each in $\Re^n$, and these are stacked as the rows of a matrix

$$\boldsymbol{X} = \begin{bmatrix} -(\boldsymbol{x}^{(1)})^T- \\ -(\boldsymbol{x}^{(2)})^T- \\ \vdots \\ -(\boldsymbol{x}^{(m)})^T- \end{bmatrix} \tag{2}$$

PCA assumes that $\boldsymbol{X}$ is an approximately low rank matrix, i.e., that $\boldsymbol{X} = \boldsymbol{L} + E$ with $||E|| \ll ||\boldsymbol{X}||$ and $r := rank(\boldsymbol{L}) \ll \min(n, m)$.

Notice each data vector is n-length. n may be very large can, e.g., can be equal to the image size if one uses all the pixels as "features". The question is can we reduce the dimension of the data without loosing too much "information"? This would help in the followin ways:

- Less storage needed to save the data.
- Analysis of the reduced-dimension dataset, e.g., regression or classification, will be faster;
- If the original data is noisy, the data analysis task, e.g., regression or classification, can also be more accurate in the reduced dimensional space. This point is discussed more in the Bias-Variance tradeoff and Learnin Theory section. When you reduce the dimension from n to an $r < n$, you increase the bias in your model. But you decrease the parameter estimation error variance. The error variance is proportional to the "noise-level" in the data, so σ_e^2 in case of linear regression.
  - You may notice this for certain real datasets with PCA implemented carefully. For simulated data, to observe this, you will have to deliberately generate noisy simulated data.
  - In some other cases, there may be zero variance along some directions and when the data covariance matrix is computed, it ends up being rank deficient. Hence it cannot be inverted, but inversion is needed for example, for GDA based classification. This is actually the perfect application where PCA resolve the problem and may in fact help improve classification accuracy.
- Finally: PCA helps to get subspace coefficients which are uncorrelated. This is needed in some applications.
  - For this, we need not reduce the dimension though, we can obtain uncorrelated variables in n dimensions too by using the full SVD.

13.2 Practical information: How to implement PCA

13.2.1 Reduced dimension, r, is specified

INPUT: $m \times n$ feature vectors' matrix $\boldsymbol{X}$, chosen rank r.

1. Compute $\hat{\mu} = \frac{1}{m}\sum_i \boldsymbol{x}^{(i)}$
2. For $i = 1, 2, \ldots, m$, let $\boldsymbol{z}^{(i)} = \boldsymbol{x}^{(i)} - \hat{\mu}$ and let

$$\boldsymbol{Z} = \begin{bmatrix} -(\boldsymbol{z}^{(1)})^T - \\ -(\boldsymbol{z}^{(2)})^T - \\ \vdots \\ -(\boldsymbol{z}^{(m)})^T - \end{bmatrix} \tag{3}$$

   OR: compute directly

$$\boldsymbol{Z} = \boldsymbol{X} - \boldsymbol{X}\mathbf{1}_n\mathbf{1}_m^T$$

   where $\mathbf{1}_n = [1\ 1\ \ldots\ 1]^T$ is an n-length vector of ones.

3. Compute the singular value decomposition (SVD) of $\boldsymbol{Z}$ and set $\boldsymbol{V}$ equal to the r right singular vectors with the largest singular values (called "top r" right singular vectors), i.e., compute

$$\boldsymbol{Z} \overset{SVD}{=} \boldsymbol{U}_{full}\boldsymbol{S}_{full}\boldsymbol{V}_{full}^T$$

   where $\boldsymbol{U}_{full}, \boldsymbol{V}_{full}$ are unitary matrices and $\boldsymbol{S}_{full}$ is an $m \times n$ diagonal matrix with non-negative entries (singular values) arranged in decreasing order of magnitude.

   Set $\boldsymbol{V} = \boldsymbol{V}_{full}(:, 1:r)$ in MATLAB notation (set $\boldsymbol{V}$ equal to first r columns of $\boldsymbol{V}_{full}$)

   Thus $\boldsymbol{V}$ is the $n \times r$ matrix whose columns span the computed *principal subspace*.

   - We can also obtain $\boldsymbol{V}$ as the eigenvectors with the r largest eigenvalues (called top r eigenvectors) of $\boldsymbol{Z}^T\boldsymbol{Z}$.

4. Project the origianl data $\boldsymbol{X}$ to $range(\boldsymbol{V})$: compute $\boldsymbol{b}^{(i)} = \boldsymbol{V}^T\boldsymbol{x}^{(i)}$ for each i or equivalently, compute

$$\boldsymbol{B} = \boldsymbol{X}\boldsymbol{V}$$

   Thus $\boldsymbol{B}$ is $m \times r$. These are the new feature vectors in the reduced dimensional space.

OUTPUT: $m \times r$ matrix $\boldsymbol{B}$: reduced dim feature vectors; $n \times r$ matrix $\boldsymbol{V}$: principal subspace.

- If the rank r approximation of $\boldsymbol{X}$ is needed, this is obtained as the $m \times n$ matrix

$$\boldsymbol{L} = \boldsymbol{B}\boldsymbol{V}^T$$

13.2.2 Deciding r

So far we have assumed that the desired lower-dimension r is given. However, there is no one correct way of deciding r in practice. Two common approaches:

1. A common heuristic is the 90% or some other high-enough percent heuristic: retain all eigenvectors so the that variance in the reduced dimensional space is at least 99% of the total variance of the data. In other words, find the smallest value r so that

$$\sum_{i=1}^{r}\sigma_i^2 \geq 0.99\sum_{i=1}^{\min(m,n)}\sigma_i^2.$$

   There is nothing "special" about 99%, we could also use another percentage.

2. An alternative approach is to pick the r that best for the final application for which PCA is being used as a pre-processing step. For example, if PCA is applied to reduce the dimension of the input training data used for linear regression, then we use cross-validation to compute the value of r that minimizes the test-MSE. See Homework 4.

Details for the second approach. Consider Linear Regression. Given $\boldsymbol{X}$, $\boldsymbol{y}$ and $\boldsymbol{X}_{test}, \boldsymbol{y}_{test}$ (split all available data into 10% testing and 90% training). OR BETTER: use cross validation (most efficient but more difficult to explain here).

1. Loop over r from 1 to n

   (a) Compute $\boldsymbol{V}$ and $\boldsymbol{B}$ using the PCA algorithm given above with inputs $\boldsymbol{X}$ and r

   (b) Compute $\hat{\theta}$ using $\boldsymbol{B}$ (instead of $\boldsymbol{X}$) and make sure to learn the bias term as well as explained earlier.

   (c) For each test data query vector, $\boldsymbol{x}_{test}^{(i)}$,

      i. compute $\boldsymbol{b}_{test}^{(i)} = \boldsymbol{V}^T \boldsymbol{x}_{test}^{(i)}$

      ii. predict $\hat{y}_{test}^{(i)} = \hat{\theta}^T \boldsymbol{b}_{test}^{(i)}$

      Alternative approach:

      i. Compute $\hat{\theta}_x = \boldsymbol{V}\hat{\theta}$,

      ii. then predict $\hat{y}_{test}^{(i)} = \hat{\theta}_x^T \boldsymbol{x}_{test}^{(i)}$

      Third alternate option: matrix-wise computation (fastest way to implement)

      i. compute $\boldsymbol{B}_{test} = \boldsymbol{X}_{test}\boldsymbol{V}^T$

      ii. obtain the vector of predictions for test data, $\hat{\boldsymbol{y}}_{test} = \boldsymbol{B}_{test}\hat{\theta}$

      I think the algebra in the above is correct – so that all three options should yield the exact same answer, but please verify to be sure.

   (d) Compute $NormalizedTestMSE(r) = ||\boldsymbol{y}_{test} - \hat{\boldsymbol{y}}_{test}||_2^2 / ||\boldsymbol{y}_{test}||_2^2$ on the test data as explained before

2. Pick r for which Test-MSE(r) is the smallest.

13.3 Why above procedure for PCA

Claim: All of the below optimization problems are solved by the PCA procedure above

13.3.1 Statistical Optimality

First assume everything is zero mean.

1. PCA finds the subspace $\boldsymbol{V}$ ($\boldsymbol{V}$ is a matrix with orthonormal columsn that define the subspace) and the projected random vector $\boldsymbol{b}$ so that the expected value of the squared 2-norm of the reconstruction error $\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b}$ is minimized. Thus, it solves

$$\min_{\boldsymbol{b}\in\Re^r, \boldsymbol{V}\in\Re^{n\times r}: \boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b}||_2^2]$$

   If we minimize over $\boldsymbol{b}$ first as a function of $\boldsymbol{V}$, then this is a standard Least Squares problem whose solution is

$$\boldsymbol{b} = (\boldsymbol{V}^T\boldsymbol{V})^{-1}\boldsymbol{V}^T\boldsymbol{x} = \boldsymbol{V}^T\boldsymbol{x} \quad \text{since } \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I} \text{ in this case}$$

   Thus, we need to solve

$$\min_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||\boldsymbol{x} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}||_2^2]$$

Since $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$, thus $||\boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}||_2^2 = ||\boldsymbol{V}^T\boldsymbol{x}||_2^2$ and so $||\boldsymbol{x} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}||_2^2 = ||\boldsymbol{x}||^2 - ||\boldsymbol{V}^T\boldsymbol{x}||^2$. With this, the above is also equivalent to

$$\max_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||\boldsymbol{V}^T\boldsymbol{x}||^2]$$

Using property of trace, this is further equivalent to

$$\max_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} trace(\boldsymbol{V}^T\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]\boldsymbol{V})$$

This is another commonly use definition for PCA: PCA finds the r directions of largest variance of the data. Here $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$ is the covariance matrix.

2. PCA can also be understood as minimizing the worst-case (largest) expected reconstruction error in any direction: for direction $\boldsymbol{w}$, this is $|\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})|$

$$\min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \max_{\boldsymbol{w}:||\boldsymbol{w}||_2=1} \mathbb{E}[(\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b}))^2]$$

Using the properties of trace, this is equivalent to

$$\min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \max_{\boldsymbol{w}:||\boldsymbol{w}||_2=1} \boldsymbol{w}^T\mathbb{E}[(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})^T]\boldsymbol{w} = \min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \lambda_{\max}(\mathbb{E}[(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})^T])$$

The last equality follows using the variational definition of the maximum eigenvalue.

In the nonzero mean case, we do either of the above for $(\boldsymbol{x} - \mu)$.

13.3.2 Optimality from a computational viewpoint

This assumes that we are given a data matrix $\boldsymbol{X}$ (or $\boldsymbol{Z}$ after subtracting the empirically computed mean). The goal is to find a rank r approximation to $\boldsymbol{X}$, denoted by $\boldsymbol{L}$, that minimizes either the Frobenius norm of the error or the induced 2-norm of the error. Thus PCA on $\boldsymbol{X}$ solves

1.

$$\min_{\boldsymbol{L} \text{ rank r}} ||\boldsymbol{X} - \boldsymbol{L}||_F^2$$

Any rank r matrix $\boldsymbol{L}$ can be expressed as $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{V}^T$ where $\boldsymbol{V}$ is an $n \times r$ matrix with orthonormal columns ($\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$) and $\boldsymbol{B}$ is $m \times r$. Thus, above is equivalent to

$$\min_{\boldsymbol{B},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} ||\boldsymbol{X} - \boldsymbol{B}\boldsymbol{V}^T||_F^2 = \min_{\boldsymbol{B},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)}||_2^2$$

The last equality follows by expressing $\boldsymbol{X}$ and $\boldsymbol{B}$ in terms of their rows.

With the above, the minimizations over the different $\boldsymbol{b}^{(i)}$s (as a function of $\boldsymbol{V}$) are decoupled and each is a simple LS problem that is solved by $\boldsymbol{b}^{(i)} = (\boldsymbol{V}^T\boldsymbol{V})^{-1}\boldsymbol{V}^T\boldsymbol{x}^{(i)} = \boldsymbol{V}^T\boldsymbol{x}^{(i)}$. Substituting this for $\boldsymbol{b}^{(i)}$, we need to solve

$$\min_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}^{(i)}||_2^2$$

2. It also solves

$$\min_{\boldsymbol{L} \text{ rank r}} ||\boldsymbol{X} - \boldsymbol{L}||_2^2 = \min_{\boldsymbol{L} \text{ rank r}} \lambda_{\max}((\boldsymbol{X} - \boldsymbol{L})^T(\boldsymbol{X} - \boldsymbol{L}))$$

Once again writing $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{V}^T$, the above is equivalent to

$$\min_{\boldsymbol{B},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \lambda_{\max}((\boldsymbol{X} - \boldsymbol{B}\boldsymbol{V}^T)^T(\boldsymbol{X} - \boldsymbol{B}\boldsymbol{V}^T))$$

Here again, expressing $\boldsymbol{X}$ and $\boldsymbol{B}$ in terms of their rows, $(\boldsymbol{X} - \boldsymbol{B}\boldsymbol{V}^T)^T(\boldsymbol{X} - \boldsymbol{B}\boldsymbol{V}^T) = \sum_{i=1}^{m}(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})^T$ and so, the above is further equivalent to

$$\min_{\boldsymbol{B},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \lambda_{\max}(\sum_{i=1}^{m}(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})^T)$$

13.4 Proofs of the above claims

First we prove that the statistical optimality claims are equivalent to the computational claims given above if we use an empirical approximation to the expected values. Next we prove that the r-SVD solution solves each of the two computational claims above.

13.4.1 Equivalence between statistical and computational claims 1

Using the empirical approximation

$$\mathbb{E}[||\boldsymbol{x} - \boldsymbol{V}V^T\boldsymbol{x}||^2] \approx \frac{1}{m}\sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}^{(i)}||_2^2$$

and thus the first claims from the statistical and computational viewpoints are equivalent, i.e.

$$\min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b}||_2^2] = \min_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||\boldsymbol{x} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}||_2^2]$$

is equivalent to

$$\min_{\boldsymbol{L} \text{ rank r}} ||\boldsymbol{X} - \boldsymbol{L}||_F^2 = \min_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}^{(i)}||_2^2$$

13.4.2 Proof of computational claim 1: Frobenius norm optimality

Goal is to find a rank r matrix $\boldsymbol{L}$ that solves

$$\min_{\boldsymbol{L}:\text{rank } r} \|\boldsymbol{X} - \boldsymbol{L}\|_F^2$$

As explained above, this is equivalent to $\min_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}^{(i)}||_2^2$. This is further equivalent to

$$\max_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{i=1}^{m} ||\boldsymbol{V}^T\boldsymbol{x}^{(i)}||_2^2$$

Using $\sum_{i=1}^{m} ||\boldsymbol{V}^T\boldsymbol{x}^{(i)}||_2^2 = ||\boldsymbol{X}\boldsymbol{V}||_F^2 = \sum_{j=1}^{r} ||\boldsymbol{X}\boldsymbol{v}_j||_2^2 = \sum_{j=1}^{r} \boldsymbol{v}_j^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_j$, the above is equivalent to

$$\max_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{j=1}^{r} \boldsymbol{v}_j^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_j$$

If we did not have the constraint $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$ we now have r decoupled problems to solve. The constraint itself can be rewritten as

$$\{\boldsymbol{V} : \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}\} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_r \text{ unit 2-norm and } \boldsymbol{v}_2 \perp \boldsymbol{v}_1,\ \boldsymbol{v}_3 \perp span(\boldsymbol{v}_1, \boldsymbol{v}_2), \text{ and so on}\}$$

To write things simply, suppose $r = 3$ so that $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3]$. Then,

$$\max_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \sum_{j=1}^{r}\sum_{i=1}^{m} \boldsymbol{v}_j^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_j$$

$$= \max_{\boldsymbol{v}_3 \perp span(\boldsymbol{v}_1,\boldsymbol{v}_2):||\boldsymbol{v}_3||_2=1} \left(\max_{\boldsymbol{v}_2 \perp \boldsymbol{v}_1:||\boldsymbol{v}_2||_2=1} \left(\left(\max_{\boldsymbol{v}_1:||\boldsymbol{v}_1||_2=1} \sum_{i=1}^{m} \boldsymbol{v}_1^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_1 \right) + \boldsymbol{v}_2^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_2 \right) + \boldsymbol{v}_3^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_3 \right)$$

Consider the max over $\boldsymbol{v}_1$. By definition of the first eigenvector (eigenvector with largest eigenvalue), the maximizer of $\max_{\boldsymbol{v}_1:||\boldsymbol{v}_1||_2=1} \boldsymbol{v}_1^T(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v}_1$ is given by the first eigenvector of $\boldsymbol{X}^T\boldsymbol{X}$. This is also the first right singular vector of $\boldsymbol{X}$.

Now consider the max over $\boldsymbol{v}_2 \perp \boldsymbol{v}_1$. This can be simplified by writing $\boldsymbol{v}_2 = (\boldsymbol{I} - \boldsymbol{v}_1\boldsymbol{v}_1^T)\boldsymbol{z}$ and maximizing over $\boldsymbol{z}$. With this, and with expressing $\boldsymbol{X}^T\boldsymbol{X}$ in terms of its EVD, it can be shown that the maximizer will be the second eigenvector (eigenvector with second largest eigenvalue). This is the second right singular vector of X.

The proof of the second eigenvector claim is not provided; need to fill this in

For max over $\boldsymbol{v}_3$, we can express $\boldsymbol{v}_3 == (\boldsymbol{I} - \boldsymbol{v}_1\boldsymbol{v}_1^T - \boldsymbol{v}_2\boldsymbol{v}_2^T)\boldsymbol{z}$ and repeat the same argument as above.

13.4.3 Equivalence between statistical and computational claims 2

Similarly, using the empirical approximation

$$\mathbb{E}[\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})^T\boldsymbol{w}] \approx \sum_{i=1}^{m} \boldsymbol{w}^T(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})^T\boldsymbol{w}$$

the second claims from the statistical and computational viewpoints are also equivalent, i.e.

$$\min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \lambda_{\max}(\mathbb{E}[(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b})^T]) = \min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \max_{\boldsymbol{w}:||\boldsymbol{w}||_2=1} \mathbb{E}[(\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{V}\boldsymbol{b}))^2]$$

This is further equivalent to

$$\min_{\boldsymbol{L} \text{ rank r}} ||\boldsymbol{X} - \boldsymbol{L}||_2^2 = \min_{\boldsymbol{B},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \lambda_{\max}(\sum_{i=1}^{m}(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})(\boldsymbol{x}^{(i)} - \boldsymbol{V}\boldsymbol{b}^{(i)})^T)$$

13.4.4 Proof of computational claim 2: induced 2 norm optimality

Goal is to find a rank r matrix $\boldsymbol{L}$ that solves

$$\min_{\boldsymbol{L}:\text{rank } r} \|\boldsymbol{X} - \boldsymbol{L}\|_2$$

By Weyl-type inequality for singular values, $\sigma_i(M) \leq \sigma_i(M_2) + \|M - M_2\|$ for any singular value i and any two matrices M, M_2.

Thus, $\|\boldsymbol{X} - \boldsymbol{L}\|_2 \geq \sigma_i(\boldsymbol{X}) - \sigma_i(\boldsymbol{L})$.

Since $\boldsymbol{L}$ is rank r, it has only r nonzero singular values. Thus $\sigma_{r+1}(\boldsymbol{L}) = 0$. And so $\|\boldsymbol{X} - \boldsymbol{L}\|_2 \geq \sigma_{r+1}(\boldsymbol{X})$. Since this is true for any rank r matrix $\boldsymbol{L}$, it is also true for the minimizer, i.e.

$$\min_{\boldsymbol{L}:\text{rank } r} \|\boldsymbol{X} - \boldsymbol{L}\|_2 \geq \sigma_{r+1}(\boldsymbol{X})$$

Now if we can find a specific matrix $\boldsymbol{L}$ for which $\|\boldsymbol{X} - \boldsymbol{L}\|_2 = \sigma_{r+1}(\boldsymbol{X})$ that will be the minimizer (since the minimum value cannot be any smaller than this).

If we let $\hat{\boldsymbol{L}} = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ (r-SVD of $\boldsymbol{X}$), then $\boldsymbol{X} - \hat{\boldsymbol{L}} = \sum_{i=r+1}^{\min(m,n)} \sigma_i u_i v_i^T$ and so $\|\boldsymbol{X} - \hat{\boldsymbol{L}}\|_2 = \sigma_{r+1}$; here $\sigma_i = \sigma_i(\boldsymbol{X})$.

13.5 PCA de-correlates the data: what does it mean

This claim depends on how the principal subspace is defined. In (statistical) theory, we are finding $\boldsymbol{V}$ and $\boldsymbol{b}$ that solves $\min_{\boldsymbol{b},\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||(\boldsymbol{x} - \mu) - \boldsymbol{V}\boldsymbol{b} + \boldsymbol{V}\boldsymbol{V}^T\mu)||_2^2]$, i.e., it minimizes the expected value of the reconstruction error. Since the minimizer over $\boldsymbol{b}$ is $\boldsymbol{b} = \boldsymbol{V}^T\boldsymbol{x}$, we are actually finding $\boldsymbol{V}$ that solves $\min_{\boldsymbol{V}:\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{I}} \mathbb{E}[||(\boldsymbol{x} - \mu) - \boldsymbol{V}\boldsymbol{V}^T(\boldsymbol{x} - \mu)||_2^2]$ and then setting $\boldsymbol{b} = \boldsymbol{V}^T\boldsymbol{x}$. If we can find this $\boldsymbol{V}$, then the computed lower dimensional r.v. $\boldsymbol{b}$ satisfies

$$\mathbb{E}[(\boldsymbol{b} - \mathbb{E}[\boldsymbol{b}])(\boldsymbol{b} - \mathbb{E}[\boldsymbol{b}])^T] \text{ is diagonal}$$

i.e.

$$\mathbb{E}[(\boldsymbol{b} - \mathbb{E}[\boldsymbol{b}])_j(\boldsymbol{b} - \mathbb{E}[\boldsymbol{b}])_k] = 0$$

for $j \neq k$.

Proof: assume everything is zero mean for ease of writing. We have $\boldsymbol{b} = \boldsymbol{V}'\boldsymbol{x}$ *so* $\boldsymbol{b}_j = \boldsymbol{v}_j'\boldsymbol{x}$*, thus,* $\mathbb{E}[\boldsymbol{b}_j\boldsymbol{b}_k] = \boldsymbol{v}_j'\mathbb{E}[\boldsymbol{x}\boldsymbol{x}']\boldsymbol{v}_k$*. As explained in previous section,* $\boldsymbol{V}$ *is the matrix of top* r *eigenvectors of* $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}']$*, i.e., that* $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}'] = \boldsymbol{V}\Sigma V' + \boldsymbol{V}_\perp\Sigma_\perp\boldsymbol{V}_\perp'$*. Hence,* $\mathbb{E}[\boldsymbol{b}_j\boldsymbol{b}_k] = \boldsymbol{v}_j'(\boldsymbol{V}\Sigma\boldsymbol{V}' + \boldsymbol{V}_\perp\Sigma_\perp\boldsymbol{V}_\perp')\boldsymbol{v}_k = \boldsymbol{v}_j'(\boldsymbol{V}\Sigma V')\boldsymbol{v}_k = 0$.

In practice, we cannot find above but only its data-based (empirical) approximation. Hence in practice, we are always finding $\boldsymbol{V}$ as the top r right singular vectors of $\boldsymbol{Z} = \boldsymbol{X} - \boldsymbol{X}\boldsymbol{1}_n\boldsymbol{1}_m^T$ (recall: where $\boldsymbol{1}_n = [1\ 1\ \ldots\ 1]^T$ is an n-length vector of ones). With this choice of $\boldsymbol{V}$, "uncorrelated" means the following:

$$\sum_{i=1}^{m}(\boldsymbol{b}^{(i)})_j(\boldsymbol{b}^{(i)})_k = 0$$

for $j \neq k$. In other words, the columns of the matrix $\boldsymbol{B}$ are mutually orthogonal.

Proof: same basic idea as above.

14 Unsupervised Learning: Clustering

Use ML-cs229- notes on Clustering to see the figures.

14.1 Problem

Given unlabeled data/features $\boldsymbol{x}^{(i)}$, $i = 1, 2, \ldots, m$, the goal is to partition the dataset into "cohesive" clusters (all points in same cluster are "close" while those in different clusters are "far"). Suppose we want to partition into k clusters. Then the goal can also be stated as: for each i, find the class label $y^{(i)}$ (this can take values from $\{1, 2, \ldots, k\}$).

14.2 k-means clustering

The goal is to find the class labels $y^{(i)}$ for each data point and the cluster centers so that the following cost is minimized

$$J(\mu_j, j = 1, 2, \ldots, k, \boldsymbol{y}) = \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \mu_{y^{(i)}}||_2^2$$

The original k-means clustering algorithm provides an Alternating-Minimization (AltMin) algorithm to minimize the above cost.

1. Initialize cluster centers $\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_k$. Can do random init.
2. Repeat
   (a) For each $i = 1, 2, \ldots, m$, find the class labels
   $$\hat{y}^{(i)} = \arg\min_{j=1,2,\ldots,k} ||\boldsymbol{x}^{(i)} - \hat{\mu}_j||_2^2.$$
   (b) Update cluster centers: for each $j = 1, 2, \ldots, k$, compute
   $$\hat{\mu}_j = \frac{\sum_{i=1}^{m} \mathbb{1}(\hat{y}^{(i)} == j)\boldsymbol{x}^{(i)}}{\sum_{i=1}^{m} \mathbb{1}(\hat{y}^{(i)} == j)}$$
   (the above is a solution to $\arg\min_{\mu_j, j=1,2,\ldots,k} \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \mu_{yhat^{(i)}}||_2^2$)

   until cluster center estimates do not change much, i.e., until $\max_j(||\hat{\mu}_j^{(t+1)} - \hat{\mu}_j^{(t)}||/||\hat{\mu}_j^{(t)}||) < threshold$ where $threshold = 0.001$ or some small fraction.

3. IMPROVED VERSION: repeat above algorithm N times with different random init's each time. For each repeat, compute the cost function value $J(\hat{\mu}_j, j = 1, 2, \ldots, k, \hat{y})$. Keep the output of the repeat with the smallest cost.

IMPROVEMENT 2: one can replace the regular Euclidean distance to the cluster center by any other distance that is more relevant to the application. As an example, if it is known that different features are likely to have significantly different variances, one could init with $\hat{\Sigma}_j = \boldsymbol{I}$, replace $\arg\min_{j=1,2,\ldots,k} ||\boldsymbol{x}^{(i)} - \hat{\mu}_j||_2$ in step 2a by

$$\arg \min_{j=1,2,\ldots,k} (\boldsymbol{x}^{(i)} - \hat{\mu}_j)\hat{\Sigma}_j^{-1}(\boldsymbol{x}^{(i)} - \hat{\mu}_j)$$

and in step 2b, update

$$\hat{\Sigma}_j = \frac{\sum_{i=1}^m \mathbb{1}(\hat{y}^{(i)} == j)(\boldsymbol{x}^{(i)} - \hat{\mu}_j)(\boldsymbol{x}^{(i)} - \hat{\mu}_j)^T}{\sum_{i=1}^m \mathbb{1}(\hat{y}^{(i)} == j)}$$

14.3 Alternating Minimization (AltMin) algorithm

AltMin is another approach to solve an optimization problem. It is a better one to use that Gradient Descent when the variable to be minimized over, $\boldsymbol{x}$, can be split into two subsets of variables $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2]$ such that minimizing over $\boldsymbol{x}_1$ keeping $\boldsymbol{x}_2$ fixed and vice versa is either closed form or otherwise easy to do.

AltMin proceeds as follows

1. Randomly initialize $\boldsymbol{x}_1$ to $\hat{\boldsymbol{x}}_1$.
2. Repeat
   (a) Minimize over $\boldsymbol{x}_2$ keeping $\boldsymbol{x}_1$ fixed at $\hat{\boldsymbol{x}}_1$, i.e., compute
   $$\hat{\boldsymbol{x}}_2 = \arg\min_{\boldsymbol{x}_2} J(\hat{\boldsymbol{x}}_1, \boldsymbol{x}_2)$$
   (b) Minimize over $\boldsymbol{x}_1$ keeping $\boldsymbol{x}_2$ fixed at $\hat{\boldsymbol{x}}_2$, i.e., compute
   $$\hat{\boldsymbol{x}}_1 = \arg\min_{\boldsymbol{x}_1} J(\boldsymbol{x}_1, \hat{\boldsymbol{x}}_2)$$
   until "convergence" i.e. estimates of $\hat{\boldsymbol{x}}_1$ do not change much from previous to current iteration.

Like Grad Desc, AltMin also only converges to the local minimum of the cost function. Thus to make it work better one can run it N times with different random init's and pick the solutions that result in the smallest cost function value.

14.4 Probabilistic Model (Generative Model) for Clustering: Gaussian Mixture Model (GMM)

The Gaussian Mixture Model or GMM is a common way to specify a clustering problem. $\boldsymbol{x}$ follows the GMM with k components means that

$$p(\boldsymbol{x}; \theta) = \sum_{j=1}^k \mathcal{N}(\boldsymbol{x}; \mu_j, \Sigma_j)\phi_j$$

where ϕ_j's are the mixture weights (probability of x coming from the j-th class in the mixture) and thus $\sum_{j=1}^k \phi_j = 1$. We often refer to j as the class labels.

GMM: means that $\boldsymbol{x}$ is generated from class j with probability ϕ_j, and given that it is generated from class j, it follows a Gaussian distribution with mean μ_j and covariance Σ_j.

The model assumed by Gaussian discriminant analysis (GDA) covered earlier and in HW 2 is also GMM. See Sec 5.2. Except there we had labeled data (for each training data point, the class label was available),

so it was easy to "learn" the model parameters. As a result, the learning of ϕ_j's was decoupled from the learning of the mean and covariances. As a result we in fact got closed form expressions for the MLE.

Here we do not have class labels and thus the learning problem is more difficult. We need to use Gradient Descent or some other iterative approach.

14.5 EM algorithm for MLE for Gaussian Mixture Model

A popular approach for MLE for the GMM is the EM algorithm.

EM algorithm is another approach (besides AltMin and Grad Descent) to solve a Maximum Likelihood estimation (MLE) problem. Like AltMin it is useful for certain types of problems in which, by using some tricks, part of the problem can be made non-iterative (closed form) or can otherwise be simplified.

Consider Maximum Likelihood Estimation over i.i.d. data samples $x^{(i)}$ coming from a distribution $p(x;\theta)$. The goal is to maximum the likelihood or equivalently its log, i.e.,

$$\max_{\theta} \ell(\theta) := \sum_{i=1}^{m} \log p(x^{(i)};\theta)$$

For simple cases like Gaussian, this is easy to do (can get a closed form expression). But in other problems it is not easy, the cost function is not convex for example. In certain such settings, the EM algorithm helps simplify. Consider the Gaussian mixture model with k components. Then

$$p(x;\theta) = \sum_{j=1}^{k} \mathcal{N}(x;\mu_j,\Sigma_j)\phi_j$$

where ϕ_j's are the mixture weights (probability of x coming from the j-th class in the mixture) and thus $\sum_{j=1}^{k}\phi_j = 1$. We often refer to j as the class labels. Thus,

$$\ell(\theta) = \sum_{i=1}^{m} \log \sum_{j=1}^{k} \phi_j \mathcal{N}(x;\mu_j,\Sigma_j) \quad \text{s.t.} \quad \sum_{j} \phi_j = 1$$

and the parameters θ are

$$\theta = \{\mu_j, \Sigma_j, \phi_j\}, j = 1, 2, \ldots k$$

with μ_j being $n \times 1$, ϕ_j is a scalar and Σ_j is $n \times n$. Recall $\mathcal{N}(\boldsymbol{x};\mu,\Sigma) := C\exp(-(\boldsymbol{x}-\mu)^T\Sigma^{-1}(\boldsymbol{x}-\mu) + \log(det(Sigma))$ where C is a constant (contains the $1/\sqrt{2\pi}$ etc terms).

$\ell(\theta)$ is a pretty messy expression to even compute the gradient of: notice it involves $\log\sum_j \phi_j \exp(....)$. And Grad Descent will not work well. Compare this with the expression we had for GDA: since y^i's were known, there was no log of sums of weighted exponentials. We instead just had sum of log of exponentials which simplified easily.

For the above problem, the following is an easier approach – somehow try to approximate the "complete data likelihood". Since class labels are not available, we call this "incomplete data". The idea proceeds as follows.

We will try to lower bound $\ell(\theta)$ and then maximize the lower bound by Alternating-Minimization. To do this we will first multiply divide the above expression by $q_i(j)$ which are such that $q_i(j) \geq 0$ and

$\sum_{j=1}^{k} q_i(j) = 1$. So one can think of them as some probability distribution over the class labels.

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{m} \log \sum_{j=1}^{k} \phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j) \\
&= \sum_{i=1}^{m} \log \sum_{j=1}^{k} \phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j) \frac{q_i(j)}{q_i(j)} \\
&= \sum_{i=1}^{m} \log \sum_{j=1}^{k} q_i(j) \frac{\phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)}{q_i(j)} \\
&\geq \sum_{i=1}^{m} \sum_{j=1}^{k} q_i(j) \log \frac{\phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)}{q_i(j)}
\end{aligned}$$

with equality holding if and only if $q_i(j) = c\phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)$. The last step used log-sum (Jensen)'s inequality. Since the $q_i(j)$'s sum to 1, we get that

$$c = \frac{1}{\sum_{j=1}^{k} \phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)}.$$

In all of above we also needed to keep the constraint $\sum_j \phi_j = 1$ but we skipped it for ease of writing.

Now we will use AltMin to MAXIMIZE the above LOWER BOUND ON $\ell(\theta)$.

By the above approach, we have increased the number of unknowns - now the unknowns include $q_i(j)$'s and θ. BUT, the maximization over θ given $q_i(j)$'s is easy (closed form, as we will see below); and the maximization over $q_i(j)$'s given θ fixed is also easy (the lower bound is maximized when the inequality holds with equality, i.e., for $q_i(j) = c\phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)$ with c specified above).

14.6 Final EM algorithm for Gaussian Mixture Models

Thus, AltMin over $q_i(j)$'s and θ is our FINAL ALGORITHM. It is called "EM algorithm for Gaussian Mixture Models". This is summarized next.

- Initialize the values of θ. Recall $\theta = \{\phi_j, \mu_j, \Sigma_j\}, j = 1, 2, ..k$. Can be random or something else.
- Iterate the following two steps until "convergence" (some reasonable stopping criterion holds):
  - E-step (maximize lower bound on $\ell(\theta)$ over $q_i(j)$'s, holding θ fixed): this is obtained by

$$q_i(j) = \frac{\phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)}{\sum_{j=1}^{k} \phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)}, \; j = 1, 2, \ldots, k, i = 1, 2, \ldots, m.$$

  - M-step (maximize lower bound on $\ell(\theta)$ over θ holding $q_i(j)$ fixed at the above value:

$$\max_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{k} q_i(j) \log \frac{\phi_j \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)}{q_i(j)}$$

    Since $q_i(j)$'s are held fixed, this is equivalent to

$$\max_{\theta} \ell_2(\theta) := \sum_{j=1}^{k} \sum_{i=1}^{m} q_i(j) [\log \phi_j + \log \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)] \;\; \text{s.t.} \;\; \sum_j \phi_j = 1$$

    Notice we have converted log of weighted sums of exponentials to "weighted sum of log of exponentials", so the form of above expression is similar to that for GDA.

    This is easy, it is almost like GDA because it can be separated out over subsets of variables:

* Solve for ϕ_j's

$$\max_{\phi_1,\phi_2,\ldots,\phi_k} \sum_{j=1}^{k} q_i(j) \log \phi_j \quad \text{s.t.} \sum_j \phi_j = 1$$

This is solved by $\hat{\phi}_j = \frac{1}{m} \sum_{i=1}^{m} q_i(j)$. This [2]

* For each $j = 1, 2, \ldots, k$,

$$\max_{\mu_j,\Sigma_j} \sum_{j=1}^{k} \sum_{i=1}^{m} q_i(j) \log \mathcal{N}(x^{(i)}; \mu_j, \Sigma_j)$$

This is solved by

$$\hat{\mu}_j = c \sum_{i=1}^{m} q_i(j) x^{(i)}$$

and

$$\hat{\Sigma}_j = c \sum_{i=1}^{m} q_i(j)(x^{(i)} - \hat{\mu}_j)(x^{(i)} - \hat{\mu}_j)^T$$

with $c = \frac{1}{\sum_{i=1}^{m} q_i(j)}$.

- Repeat with N different initializations. Pick the one for which we get the largest value of $\ell(\theta)$

The "E step" is the "Expectation step". Reason it is called this is the following. We have data that is incomplete. We do not know the values of the misisng entries, and so we do not know the complete data likelihood (in this case this corresponds to the likelihood func for GDA). But given estimates of the paramters from the previous iteration, we can compute what is called the posterior expectation of the complete data likelihood.

- The $q_i(j)$'s are called the posterior probabilities of $y^{(i)} = j$ given observed data, and
- $\ell_2(\theta)$ is called the posterior Expectation of the complete data likelihood.

Thus "E step" computes this posterior expectation given previous estimate of parameters, while "M step" maximizes this posterior expectation to find a new value of the parameters. Hence the same EM algorithm.

14.7 General EM

In the above writing, I have written the above out for the Gaussian Mixture Model first to make it easier to understand. More generally, you could introduce any "missing data" variables $y^{(i)}$ and use a process similar to the above. If the labels are discrete-valued, then we can write things as

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{m} \log \sum_{j=1}^{k} p(x^{(i)}, y^{(i)} = j; \theta) \quad \text{s.t.} \sum_{j=1}^{k} q_i(y^{(i)} = j) = 1 \\
&= \sum_{i=1}^{m} \log \sum_{j=1}^{k} p(x^{(i)}, y^{(i)} = j; \theta) \frac{q_i(y^{(i)} = j)}{q_i(y^{(i)} = j)} \quad \text{s.t.} \sum_{j=1}^{k} q_i(y^{(i)} = j) = 1 \\
&= \sum_{i=1}^{m} \log \sum_{j=1}^{k} q_i(y^{(i)} = j) \frac{p(x^{(i)}, y^{(i)} = j; \theta)}{q_i(y^{(i)} = j)} \quad \text{s.t.} \sum_{j=1}^{k} q_i(y^{(i)} = j) = 1 \\
&\geq \sum_{i=1}^{m} \sum_{j=1}^{k} q_i(y^{(i)} = j) \log \frac{p(x^{(i)}, y^{(i)} = j; \theta)}{q_i(y^{(i)} = j)} \quad \text{s.t.} \sum_{j=1}^{k} q_i(y^{(i)} = j) = 1
\end{aligned}$$

[2] we actually need $c = \frac{1}{\sum_{j=1}^{k} \sum_{i=1}^{m} q_i(j)}$; not hard to see that this simplifies to $1/m$.

We now use Alternating-Maximization to maximize the above lower bound over $q_i(y^{(i)} = j)$'s keeping θ fixed and vice-versa.

We know that the RHS cannot larger than its upper bound, thus if we can find a value that helps achieve the upper bound we are done. The following is easy to see, if $\sum_j g_j = 1$, then, $\sum_j g_j \log c = \log c = \log \sum_j g_j c$. Thus the lower bound is maximized when

$$q_i(y^{(i)} = j) = cp(x^{(i)}, y^{(i)} = j; \theta)$$

Since the above sums to one over j, we need $c = 1/p(x^i; \theta)$ and so

$$q_i(y^{(i)} = j) = p(y^{(i)} = j | x^{(i)}; \theta)$$

Thus the general EM algorithm proceeds as follows.

- E step: keeping θ fixed at its previous value, compute

$$q_i(y^{(i)} = j) = p(y^{(i)} = j | x^{(i)}; \theta), j = 1, 2, \ldots, k, i = 1, 2, \ldots, m$$

- M step: keeping $q_i(y^{(i)} = j)$ fixed at above value, compute

$$\begin{aligned}
&\arg\max_\theta \sum_{i=1}^{m} \sum_{j=1}^{k} q_i(y^{(i)} = j) \log \frac{p(x^{(i)}, y^{(i)} = j; \theta)}{q_i(y^{(i)} = j)} \\
&= \arg\max_\theta \sum_{i=1}^{m} \sum_{j=1}^{k} q_i(y^{(i)} = j) \log p(x^{(i)}, y^{(i)} = j; \theta) \\
&= \arg\max_\theta \sum_{j=1}^{k} \sum_{i=1}^{m} q_i(y^{(i)} = j) \log p(x^{(i)}, y^{(i)} = j; \theta)
\end{aligned}$$

  This is often easier if the parameters θ have a mixture-model type form and the maximization can be separated out

In more general settings $y^{(i)}$ may be real-valued (continuous r.v.'s). In these cases, the summation over j gets replaced by integration, but a lot of the essential approach remains the same. A log-sum inequality exists for integrals also.

15 Deep Learning / Neural Networks: basic idea and training

This is another Supervised Learning approach.

The simplest neural net is the Feed-forward Network also called the Multilayer Perceptron or MLP. Each neuron receives a weighted sum of the outputs of the neurons of the previous layer, and applies a nonlinear "activation function" on this. Thus neuron j in layer k receives

$$z_j^k = \sum_{i=1}^{r_k} w_{ij}^k o_i^{k-1}$$

as input and outputs

$$o_j^k = g(z_j^k)$$

Here $g(z)$ is an element-wise nonlinearity. It could be the sigmoid function $1/(1 + e^{-z})$ or the Rectified Linear Unit (ReLU) function $\max(z, 0)$ or the tanh function.

Vectorizing the above, the NN can be expressed as

$$\boldsymbol{o}^k = g_{vec}(\boldsymbol{z}^k), \quad \boldsymbol{z}^k = \boldsymbol{W}^k \boldsymbol{o}^{k-1}$$

or equivalently,

$$\boldsymbol{z}^k = \boldsymbol{W}^k g_{vec}(\boldsymbol{z}^{k-1})$$

Here $g_{vec}(\boldsymbol{z})$ applies $g(.)$ to each entry of the vector $\boldsymbol{z}$.

The first layer takes the input $\boldsymbol{x}$ as input thus

$$\boldsymbol{z}^1 = \boldsymbol{x}$$

Suppose the NN has 10 layers. The final (output) layer has only one neuron which outputs

$$\hat{y} = g(\boldsymbol{z}_1^{10})$$

Thus, the NN is

$$\hat{y} = g(\boldsymbol{W}^{10} g_{vec}(\boldsymbol{W}^9 g_{vec}(\boldsymbol{W}^8 \dots g_{vec}(\boldsymbol{x}))))$$

where $\boldsymbol{W}^{10}$ is a row vector (instead of a matrix).

15.1 Training: Back-propagation

Given training data $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, 2, \dots, m$, we use gradient descent or stochastic / mini-batch GD to train. For all of these, the first task is to define the cost function $E(\hat{y}(\boldsymbol{x}), y)$ and to compute its gradient w.r.t. to each weight in each layer, i.e., compute

$$\frac{\partial E}{\partial w_{ij}^k}$$

This computation requires careful application of chain rule of differentiation. This leads to the following algorithm: consider a 10 layer NN and let r_k denote the number of neurons in layer k

- For a given input $\boldsymbol{x}$, compute the outputs of all the layers and $\hat{y}$.
- Compute the following intermediate quantity:

  $$\delta_i^k := \frac{\partial E}{\partial z_i^k}$$

  using the following backward recursion (back-propagation)

  - compute

    $$\delta_1^{10} = \frac{\partial E}{\partial \hat{y}}(\hat{y}, y) \cdot g'(z_1^{10})$$

    here $g'(z) = \frac{\partial g(z)}{\partial z}$
  - for each $k = 9, 8, \dots 1$, compute the following for each $i = 1, 2, \dots, r_k$:

    $$\delta_i^k = g'(z_i^k) \sum_{j=1}^{r_{k+1}} w_{ij}^{k+1} \delta_j^{k+1}$$

    vectorized computation in MATLAB (or do similar in Python): $\boldsymbol{\delta}^k = g'(\boldsymbol{z}^k) .* (\boldsymbol{W}^{k+1} \boldsymbol{\delta}^{k+1})$
- Compute

  $$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k \cdot o_i^{k-1}$$

  this can be vectorized too.

The above gives us $\frac{\partial E}{\partial w_{ij}^k}(\hat{y}(\boldsymbol{x}), y)$ for one input-output pair $\boldsymbol{x}, y$. The gradient w.r.t. the cost function that uses all the training data is thus

$$\frac{1}{m} \sum_{i=1}^{m} \frac{\partial E}{\partial w_{ij}^k}(\hat{y}(\boldsymbol{x}^{(i)}), y^{(i)})$$

15.2 Convolutional Neural Network (CNN or ConvNet): basic idea

To Do

15.3 Recurrent Neural Network (RNN): basic idea

To Do

15.4 Different NN architectures and when to use each

see the other handout NeuralNets-intro.pdf

16 Bias-Variance Tradeoff

16.1 What is it?

Consider a generative model: suppose that $y, \boldsymbol{x}$ satisfy

$$y = f(\boldsymbol{x}) + e,\ \mathbb{E}[e] = 0, \mathbb{E}[e^2] = \sigma^2$$

with e being zero mean "modeling error"/"noise" that is independent of $\boldsymbol{x}$, it is also independent for each data point y_i.

We do not know $f(.)$.

We try to "model" it as $\hat{y} = \hat{f}(x) = h_{\hat{\theta}}(\boldsymbol{x})$, e.g., in linear regression, $\hat{f}(x) = h_{\hat{\theta}}(\boldsymbol{x}) = \hat{\theta}^T \boldsymbol{x}$ with $\hat{\theta}$ estimated by Maximum Likelihood estimation (MLE) as described earlier using training data

$$(y^i, \boldsymbol{x}^i), i = 1, 2, \ldots, m$$

In logistic regression, $\hat{f}(x) = h_{\hat{\theta}}(\boldsymbol{x}) = g(\hat{\theta}^T \boldsymbol{x})$ with $g(.)$ being the sigmoid function. MLE uses "training data"

The question is how good is my learnt model (in terms of mean squared error on test data), i.e., for a test query $\boldsymbol{x}$, what is $\mathbb{E}[(y - \hat{f}(\boldsymbol{x}))^2]$ and what can we do to improve it?

We define Test-MSE as

$$\text{Test-MSE} := \mathbb{E}[(y - \hat{y})^2] = \mathbb{E}[(y - \hat{f}(\boldsymbol{x}))^2] = \mathbb{E}[(f(\boldsymbol{x}) + e - \hat{f}(\boldsymbol{x}))^2]$$

over test data, i.e., $\mathbb{E}[.] \equiv \mathbb{E}_{\text{test data}}[.]$. Usually $\boldsymbol{x}$ is treated as a constant, so then the expected value is over the distribution of the noise e.

Since e is test-data noise, it is independent of $\hat{f}(\boldsymbol{x}) = h_{\hat{\theta}}(\boldsymbol{x})$ since $\hat{\theta}$ was estimated using training data. Also, by assumption, e it is independent of $f(x)$. Thus, we have

$$\begin{aligned}
\text{Test-MSE} &= \mathbb{E}[e^2] + \mathbb{E}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2] \\
&= \mathbb{E}[e^2] + (\mathbb{E}[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})])^2 + Variance[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})] \\
&= \sigma^2 + Bias^2 + Variance
\end{aligned}$$

where $Variance(Z) := E[(Z - E[Z])^2]$. The first term, σ^2, depends on how noisy the data is. The second two terms depend on the "assumed model" and how well its parameters are estimated.

16.2 Bias-Variance Tradeoff for Linear Regression

THERE IS A MISTAKE HERE: NEED TO FIX THESE NOTES to all for a constant "mean" term in linear regression.

Suppose that y truly satisfies the following model

$$y = f(x) + e, \quad f(x) := {\theta_{\text{full}}}^T \boldsymbol{x}_{\text{full}},\ \mathbb{E}[e] = 0,\ Var[e] = \sigma^2$$

where x_{full} is the full n_{full} length "feature" or data. Also, the noise e is i.i.d. across various samples.

For the sake of tractability (reducing variance), when "modeling" y, we throw away some of the features to get an n-length "feature" vector $\boldsymbol{x}$; and refer to the dropped part of $\boldsymbol{x}_{\text{full}}$ as $\boldsymbol{x}_{\text{drop}}$. This is a n_{drop} length vector. Thus $n_{\text{full}} = n + n_{\text{drop}}$.

$$x_{\text{full}}^T = [\boldsymbol{x}^T, \boldsymbol{x}_{\text{drop}}]$$

Thus, y can be rewritten as

$$y = \theta^T x + \underbrace{\theta_{\text{drop}}^T \boldsymbol{x}_{\text{drop}}}_{\mu} + e.$$

Thus, the "linear regression" model for y is $\theta^T x$. Assume as before we are given training data $\{y^{(i)}, \boldsymbol{x}^{(i)}\}, i = 1, 2, \ldots, m$, and define the matrix $\boldsymbol{X}$ and the vector $\boldsymbol{y}$ as before.

We use MLE under this model to get the MLE estimate

$$\hat{\theta} := (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \text{ with } \boldsymbol{y} := \boldsymbol{X}\theta + \boldsymbol{X}_{\text{drop}}\theta_{\text{drop}} + \boldsymbol{e}$$

where $\boldsymbol{X}_{\text{drop}} := [\boldsymbol{x}_{\text{drop}}^{(1)}; \boldsymbol{x}_{\text{drop}}^{(2)}; \ldots, \boldsymbol{x}_{\text{drop}}^{(m)}]^T$ is an $m \times n_{\text{drop}}$ matrix with the dropped parts of each training data vector as its rows. Recall that $\boldsymbol{e}$ (training data noise vector) is independent of $\boldsymbol{X}$ (training data).

Thus, for a query $\boldsymbol{x}_{\text{full}}^{\text{tst}}$, with extracted features $\boldsymbol{x}^{\text{tst}}$, we predict

$$\hat{y}^{\text{tst}} = \boldsymbol{x}^{\text{tst}T}\hat{\theta}$$

The true output, y^{tst}, for the query satisfies

$$y^{\text{tst}} = \theta^T\boldsymbol{x}^{\text{tst}} + \theta_{\text{drop}}^T\boldsymbol{x}_{\text{drop}}^{\text{tst}} + e^{\text{tst}}$$

Observe that $\hat{\theta}$ satisfies

$$\hat{\theta} := (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T(\boldsymbol{X}\theta + \boldsymbol{X}_{\text{drop}}\theta_{\text{drop}} + \boldsymbol{e}) = \theta + (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}_{\text{drop}}\theta_{\text{drop}} + (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{e}$$

So,

$$\begin{aligned}
\hat{y}^{\text{tst}} - \boldsymbol{y}^{\text{tst}} &= \boldsymbol{x}^{\text{tst}T}\hat{\theta} - \boldsymbol{x}^{\text{tst}T}\theta - \boldsymbol{x}_{\text{drop}}^{\text{tst}\ T}\theta_{\text{drop}} - e^{\text{tst}} \\
&= \underbrace{\boldsymbol{x}^{\text{tst}T}\left((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}_{\text{drop}}\theta_{\text{drop}} + (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{e}\right) - \boldsymbol{x}_{\text{drop}}^{\text{tst}\ T}\theta_{\text{drop}}}_{Z} - e^{\text{tst}} \\
&= Z - e_{\text{tst}}
\end{aligned}$$

Recall that $\mathbb{E}[.]$ is expected value over test-data noise e^{tst}. As before (since e^{tst} independent of everything in Z),

$$\begin{aligned}
TestMSE := \mathbb{E}[(\hat{y}^{\text{tst}} - \boldsymbol{y}^{\text{tst}})^2] &= \mathbb{E}[(e^{\text{tst}})^2] + \mathbb{E}[Z^2] \\
&= \mathbb{E}[(e^{\text{tst}})^2] + Bias(Z)^2 + Variance(Z)
\end{aligned}$$

with

$$\begin{aligned}
Bias(Z) &:= \mathbb{E}[\boldsymbol{x}^{\text{tst}T}\left((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}_{\text{drop}}\theta_{\text{drop}} + (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{e}\right) - \boldsymbol{x}_{\text{drop}}^{\text{tst}\ T}\theta_{\text{drop}}] \\
&= \boldsymbol{x}^{\text{tst}T}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}_{\text{drop}}\theta_{\text{drop}} - \boldsymbol{x}_{\text{drop}}^{\text{tst}\ T}\theta_{\text{drop}} \\
&= [\boldsymbol{x}^{\text{tst}T}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}_{\text{drop}} - \boldsymbol{x}_{\text{drop}}^{\text{tst}\ T}]\theta_{\text{drop}}
\end{aligned}$$

The second row follows because $\boldsymbol{e}$ is independent of $\boldsymbol{X}, \boldsymbol{X}_{\text{drop}}$ (training data noise is independent of training data features) and $\boldsymbol{e}$ is independent of $\boldsymbol{x}^{\text{tst}}$ and $\boldsymbol{e}$ is zero mean.

Since the training data noise vector $\boldsymbol{e}$ is zero mean and i.i.d., we have $\mathbb{E}[\boldsymbol{e}\boldsymbol{e}^T] = \sigma^2 I$. Using this,

$$\begin{aligned} Variance(Z) &:= \mathbb{E}[(Z - Bias(Z))^2] \\ &= \mathbb{E}[(\boldsymbol{x}^{\text{tst}T}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{e})^2] \\ &= \mathbb{E}[(\boldsymbol{x}^{\text{tst}T}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{e})(\boldsymbol{e}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}^{\text{tst}})] \\ &= \boldsymbol{x}^{\text{tst}T}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbb{E}[\boldsymbol{e}\boldsymbol{e}^T]\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}^{\text{tst}} \\ &= \boldsymbol{x}^{\text{tst}T}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}^{\text{tst}}\sigma^2 \end{aligned}$$

This is upper bounded by

$$Variance(Z) \leq \frac{\|\boldsymbol{x}^{\text{tst}}\|^2}{\lambda_{\min}(\boldsymbol{X}^T\boldsymbol{X})}\sigma^2$$

1. Variance: Thus the variance is smaller if the minimum eigenvalue of $(\boldsymbol{X}^T\boldsymbol{X})$ is larger.

   - Notice that $\boldsymbol{X}$ is an $m \times n$ matrix. Consider $\boldsymbol{X}^T\boldsymbol{X}$ which is $n \times n$. If $m < n$, its minimum eigenvalue is zero. For $m > n$, its minimum eigenvalue is proportional to m/n. The reason is, for a fixed m, by the interlacing theorem for eigenvalues, as you increase n, the smallest eigenvalue can only decrease [3].

   Thus for a fixed available amount of training data, m, the Variance will either be reduced, or stay the same, if we use a smaller n.

2. Bias: Consider the Bias. This is some complicated function of θ_{drop}. We can simplify it by making more assumptions, but that is not needed. What is clear is that Bias will become larger as n_{drop} is made larger (or equivalently, as $n = n_{\text{full}} - n_{\text{drop}}$ is made smaller). Thus, as we reduce n, we increase n_{drop}, and so we increase Bias.

Thus with reducing n, Variance decreases but Bias increases.

17 Bias-Variance Tradeoff Practical Issues

As noted in previous section,

$$\text{Test-MSE} := \mathbb{E}[(y - \hat{y})^2] = \mathbb{E}[(y - \hat{f}(\boldsymbol{x}))^2] = \mathbb{E}[(f(\boldsymbol{x}) + e - \hat{f}(\boldsymbol{x}))^2]$$

over test data, i.e., $\mathbb{E}[.] \equiv \mathbb{E}_{\text{test data}}[.]$. Usually $\boldsymbol{x}$ is treated as a constant, so then the expected value is over the distribution of the noise e.

Since e is test-data noise, it is independent of $\hat{f}(\boldsymbol{x}) = h_{\hat{\theta}}(\boldsymbol{x})$ since $\hat{\theta}$ was estimated using training data. Also, by assumption, e it is independent of $f(x)$. Thus, we have

$$\begin{aligned} \text{Test-MSE} &= \mathbb{E}[e^2] + \mathbb{E}[(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2] \\ &= \mathbb{E}[e^2] + (\mathbb{E}[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})])^2 + Variance[f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})] \\ &= \sigma^2 + Bias^2 + Variance \end{aligned}$$

where $Variance(Z) := E[(Z - E[Z])^2]$. The first term, σ^2, depends on how noisy the data is. The second two terms depend on the "assumed model" and how well its parameters are estimated.

[3] To understand this with a concrete example, if the training data vectors were i.i.d. standard Gaussian, then with high probability, the minimum eigenvalue is lower bounded by $(\sqrt{m} - C\sqrt{n})^2$. Reference: Vershynin tutorial.

17.1 Approximating Test-MSE in practice

While we can write things as above, it is *not* actually possible to compute the above decomposition for test data.

Simple approach: All one can do is the following: for a given model, one can approximate Test-MSE using the following approach

- Split available training data into training and test data: in other words do not use all m data points to train, split them so that $m = m_{train} + m_{test}$.
- Use the m_{train} data points to train, i.e. to estimate θ for the assumed model
- Approximate Test-MSE by $\frac{1}{m_{test}} \sum_{j=1}^{m_{test}} (y_j - h_{\hat{\theta}}(\boldsymbol{x}_j))^2$

The above is one way to do what is called "Cross-Validation". Typically, one uses $m_{test} = 0.25m$ and $m_{train} = 0.75m$ or similar.

Leave-one-out Cross-Validation: Do above with $m_{test} = 1$, and $m_{train} = m - 1$, but repeat the procedure m times to compute the average error (Test-MSE). See Cross Validation section earlier.

17.2 How to reduce test data MSE

Try to reduce variance, while hopefully not increasing the bias too much:

- Regularization on θ: use domain knowledge such as assuming entries of θ are in decreasing order of magnitude. This was done in HW1b.
- Regularization on input features' matrix $\boldsymbol{X}$: suppose we know that $\boldsymbol{X}$ is exactly or approximately low rank. Use PCA on the feature data-set available for training followed by using the reduced dimensional "features" for the regression (or other task). Pick the reduced dimension r carefully – see PCA section. Also see HW 4, HW 5.
  - In case of Clustering (say assuming Gaussian Mixture Model) reducing k (number of classes) will reduce the variance, increasing it will reduce the bias.
- Regularization on θ – assume something weaker, e.g., model θ as being sparse, this is a generalization of the assumption used in item 1. Add $\|\theta\|_1$ into the cost function for learning θ.
- Regularization on θ – suppose prior knowledge is available that θ is close to a given vector θ_0, then add the $\|\theta - \theta_0\|_2^2$ into the cost function for finding θ.
- Naive Bayes assumption described earlier is another way to reduce the number of parameters d (this is used in settings where d is different from the feature vector length n), and hence the variance in the parameter estimates.

With each new intervention, compute the Test-MSE as explained above, see if it gets reduced or not. What is expected is this: if we increase n starting at $n = 0$, the bias will reduce significantly up to a certain value of n (and variance will not increase too much), so that Test-MSE will reduce. After a certain point, variance will start increasing significantly compared to the further reduction in bias. This is the point to stop.

So the solution can be: keep increasing n until Test-MSE begins to decrease.

When using PCA for dimension reduction, n gets replaced by r and $\boldsymbol{X}$ by $\boldsymbol{B}$ which is $m \times r$.

18 Learning Theory

based on Andrew Ng's cs229-notes-4

Hypothesis refers to a hypothesized model on the input output data. So suppose we assume that $y = sign(\theta^T \boldsymbol{x}))$ then $sign(\theta^T \boldsymbol{x}))$ is the hypothesis.

Hypothesis class $\mathcal{H}$ is the set of all hypothesis from a certain class, e.g., set of all linear classifiers is

$$\mathcal{H} = \{h : h = sign(\theta^T \boldsymbol{x}),\ \forall\ \theta \in \Re^n\}$$

Empirical Risk (Training Error) is denoted by $\hat{\varepsilon}(h)$. For a loss function $loss(y, \hat{y})$, it is computed as

$$\hat{\varepsilon}(h) = \frac{1}{m} \sum_{i=1}^{m} loss(y^i, h(\boldsymbol{x}^i))$$

Generalization Error (Test data Error) is denoted by $\varepsilon(h)$. It is the expected loss for a query sample

$$\varepsilon(h) = \mathbb{E}[loss(y, h(\boldsymbol{x}))]$$

Assumptions

- Training and Test data are generated from the same probability distribution
- All training samples as well as test sample are mutually independent. The two assumptions combined mean that all training and test data are i.i.d. (independent identically distributed).

Under the above assumptions, by using a law of large numbers' result, it can usually be argued that $\hat{\varepsilon}(h)$ converges to $\varepsilon(h)$ in probability as m goes to infinity. We will look at a simple zero-one loss function and actually work this out.

Empirical Risk Minimization (ERM) means minimize the empirical risk over all hypotheses from a certain class, i.e., try to find

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$$

Ideally what we would like to find is the hypothesis that minimizes the generalization error

$$h^* = \arg\min_{h \in \mathcal{H}} \varepsilon(h)$$

Thus the minimum generalization error is $\varepsilon(h^*)$, i.e.,

$$\varepsilon(h^*) = \min_{h \in \mathcal{H}} \varepsilon(h)$$

We would like to use h^* but we cannot compute it. We instead use $\hat{h}$ on the test data too. The question is how much worse is this? I.e., how much worse is $\varepsilon(\hat{h})$ compared to $\varepsilon(h^*) = \min_{h \in \mathcal{H}} \varepsilon(h)$? We work this out for the zero-one loss.

18.1 Two probability results we use

Lemma 18.1 (Hoeffding inequality for Bernoulli r.v.'s)**.** *Let $Z_1, Z_2, \ldots Z_m$ be iid Bernoulli random variables with parameter ϕ. Let $\hat{\phi} := \frac{1}{m} \sum_{i=1}^{m} Z_i$ be the empirical mean of these random variables. Then*

$$\Pr(|\hat{\phi} - \phi| > \gamma) \leq 2e^{-2\gamma^2 m}$$

Lemma 18.2 (Union Bound)**.** *For any K events $A_1, A_2, \ldots, A_K$*

$$\Pr(A_1 \cup A_2 \cdots \cup A_K) \leq \sum_{k=1}^{K} \Pr(A_k)$$

and thus

$$\Pr(A_1^c \cap A_2^c \cdots \cap A_K^c) \geq 1 - \sum_{k=1}^{K} \Pr(A_k)$$

18.2 Misclassification (zero-one) loss

Let us specialize to the misclassification (zero-one) loss:

$$loss(y, \hat{y}) = \mathbb{1}(y \neq \hat{y}) = \mathbb{1}(y \neq h(\boldsymbol{x}))$$

For this, the empirical risk for a given h is

$$\hat{\varepsilon}(h) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(y^i \neq h(\boldsymbol{x}^i))$$

while

$$\varepsilon(h) = \mathbb{E}[\mathbb{1}(y \neq h(\boldsymbol{x}))] = \Pr(y \neq h(\boldsymbol{x}))$$

Define the r.v. $Z_i = \mathbb{1}(y^i \neq h(\boldsymbol{x}^i))$. Clearly the Z_i are Bernoulli with probability of one equal to $\varepsilon(h)$. We will use the Hoeffding inequality

Using Hoeffding inequality, this means that, for a given hypothesis h,

$$\Pr(|\hat{\varepsilon}(h) - \varepsilon(h)| > \gamma) \leq 2e^{-2\gamma^2 m}$$

Suppose for a moment that the size of the hypothesis class is k, i.e.,

$$|\mathcal{H}| = k.$$

Then, by Union Bound,

$$\Pr(|\hat{\varepsilon}(h) - \varepsilon(h)| > \gamma, \text{ for some } h \in \mathcal{H}) \leq 2ke^{-2\gamma^2 m}$$

or equivalently

$$\Pr(|\hat{\varepsilon}(h) - \varepsilon(h)| < \gamma, \text{ for ALL } h \in \mathcal{H}) \geq 1 - 2ke^{-2\gamma^2 m}$$

For this probability to be at least $1 - \delta$, we need to set $\gamma = 2\sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$. With this we can rewrite things as follows

With probability at least $1 - \delta$, for all $h \in \mathcal{H}$, with $k = |\mathcal{H}|$,

$$|\hat{\varepsilon}(h) - \varepsilon(h)| \leq \gamma := \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

Since above is true for all h, it is true for $\hat{h}$ and h^* too. So with the above probability,

$$\varepsilon(\hat{h}) \leq \hat{\varepsilon}(\hat{h}) + \gamma \leq \hat{\varepsilon}(h^*) + \gamma \leq \varepsilon(h^*) + \gamma + \gamma = \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$$

The first and third inequalities used the Hoeffding bound from above; the second used the fact that $\hat{\varepsilon}(\hat{h})$ is the minimum over all h.

Thus, substituting for γ, we can write the following theorem

Theorem 18.3. *Consider a hypothesis class $\mathcal{H}$ with $|\mathcal{H}| = k$. And consider zero-one loss. Suppose that $\hat{h}$ minimizes the empirical risk.*

With probability $\geq 1 - \delta$,

$$\varepsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}} \varepsilon(h) \right) + 2\sqrt{\frac{1}{2m} \log(\frac{2k}{\delta})}$$

?? add corollary for lower bound on m.

This theorem works for a finite hypothesis class, but not for an infinite one, for example it does not work for the class of all linear classifiers. However since everything is done on a computer (say one on which each real number is represented by 64 bits), we can assume that there are really only a finite number of degrees

of freedom. Consider a hypothesis class with d parameters all of which are real numbers. Then the total number of possible options for hypotheses is

$$k = \underbrace{2^{64} * 2^{64} \cdots * 2^{64}}_{d \text{ times}} = 2^{64d}$$

We have the following corollary

Corollary 18.4. *Consider a hypothesis class $\mathcal{H}_d$ with d real-valued parameters. Assume implementation on a 64-bit computer so that $k = |\mathcal{H}| = 2^{64d}$. Consider zero-one loss. Suppose that $\hat{h}$ minimizes the empirical risk. Also set $\delta = 2^{-d}$. Then, we can claim the following*

With probability $\geq 1 - 2^{-d}$,

$$\varepsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}_d} \varepsilon(h) \right) + C\sqrt{\frac{65d+1}{2m}}$$

Remark 18.5. *In most of this class, we have assumed $d = n + 1$ parameters. This is true for linear regression and logistic regression. For GDA, $d > n$.*

18.3 Tradeoffs

First note: we are discussing these tradeoffs using an "upper bound" on $\varepsilon(\hat{h})$. This upper bound may or may not be tight. The discussion is valid only when it is tight.

If we increase d (for a fixed m),

- the size of the hypotheses class on a 64-bit computer is $|\mathcal{H}_d| = 2^{64d}$ as argued above. Thus, increasing d, increases its size. This means, in the first term from the above corollary, $\min_{h \in \mathcal{H}_d} \varepsilon(h)$, we are taking a minimum over a larger set. Minimizing over a larger set implies the minimum value is the same or smaller (cannot be larger). Thus the first term can only *decrease* or *stay the same* if we increase d. This term does not depend on m at all.
- but the second term, $\sqrt{\frac{65d+1}{2m}}$ clearly increases linearly with d

Increasing d and hence the size of the hypotheses class is analogous to *reducing the bias*. Typically, up to a certain increase in d, the first term will decrease (bias will decrease). But when m is fixed, this also *increases the variance*.

In fact at the very least, we need $m > d$ to even just get the RHS to be smaller than one. Notice $\varepsilon(h)$ is a probability so has to be less than one,

18.4 Connections to what we have learnt

In case of the 0-1 loss, the empirical risk is computationally not possible to minimize because one will literally have to check all 2^{64d} possible hypotheses values and compute the empirical risk for each of them to find the minimize. It is what is called a "combinatorial optimization" problem and cannot be done in any reasonable time.

Also, as noted above, we are discussing these tradeoffs using an "upper bound" on $\varepsilon(\hat{h})$. This upper bound may or may not be tight. The discussion is valid only when it is tight.

For the above two reasons, the Tradeoffs' discussion does not apply to Generative Learning Models like Gaussian Discriminant Analysis where are placing many more assumptions on our data. For instance, in GDA, once we impose $\Sigma_0 = \Sigma_1 = \Sigma$ and Σ diagonal, we have basically decoupled the learning of each scalar parameter for each feature. In this case, as seen in your Home it is possible to get good parameter estimates and thus good classification even with $m < d$, in this case $d = 2n + n + 1$.

It "somewhat" applies to SVMs, though not directly since there we aren't trying to find a $\boldsymbol{w}, b$ to minimize empirical risk (training error), instead we are trying to find w, b to maximize the worst-case margin.

18.5 VC dimension

We will not talk about VC dimension (last two pages of Andew Ng's cs229 notes-4) in this course.

Thoughts on future teaching

General introduction, Introduce Python

Python recitations' - videos and files from 2021 (by Praneeth).

Linear Regression, PCA, Bias-Variance Tradeoff

Allow Homework in Python or MATLAB .

Logistic Regression, GDA, Learning Theory.

Project for Spring 2021

Project: combine ideas of 2-4 homeworks, and work on them in more detail. First report due April 22 for comments from me or Praneeth. Final project due: last day of finals' week.

Groups: groups of at most 3. Groups of 2 required.

– Develop your own approach to write/combine code. In the report: write pseudo-code to explain exactly what you coded. What you say in the report should match up with what you coded in.

– Derive your own conclusions - first based on simulated data (figure out the correct way to generate it and also to explain it in your report) - then based on real data. Report: Explain how to generate the simulated data and why this way.

– Implement on one more dataset other than the ones we have provided in the HWs. Explain the pre-processing steps to use the dataset first, explain what you observe on real data and why. Set up your own reasonable error metrics or use existing ones. In Report: Explain the dataset, explain what you observe and why.

– Basically the report should tell me what all you did, how you coded it in (pseudo-code, not code attached), and what you observed, first for simulated, then for real data.

Examples:

- Various pre-processing steps for Linear Regression : use of PCA versus use of sparsity on theta versus use of both (first do PCA –pick r as done in HW 4 – then, on this reduced dimensional data, try to fit a sparse θ). So compare: (i) just PCA, (ii) just sparse prior on θ, (iii) combination of both.
- PCA as pre-processing step for linear regression, logistic regression and GDA (HW 4, 5)
- Various classification approaches: compare and contrast (HW 2, 3)
- PCA for classification and clustering (HW 5, 6)
- Welcome to also take any one HW and extend it beyond the existing HWs.

Deadlines:

- April 10: submit an "abstract" (¡ 1 page summary of what you will do)
- April 29: draft version of the project (if you want comments from me): strongly encouraged.
- May 5: Project due. Submit: (1) a PDF report, (2) code

Strongly encourage you to use LateX – free, best quality reports, easy to use once you figure out the basics. `https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`

Lesson Plans

The following was the lesson plan for Spring 2020 (first pandemic year).

18.6 Part 1: before March 25 (roughly)

1. Introduction to Linear Regression, Python, and the idea of Simulating Data to Test your algorithm ideas.

2. Background Mathematics material:

   - Linear Algebra (Ng's notes and a few topics from my linearalgebranotes.pdf notes): need to know SVD, eigenvalue decomposition, and Weyl's ineqiality to understand PCA carefully.

- Probability (my EE 322 notes are sufficient): needed to understand Maximum Likelihood estimation, and Generative Learning algorithms
- Basic Optimization: gradient descent (GD) and why it works, stochastic GD. Need more details to understand SVM dual program derivations (but those were skipped this year). See SummaryNotes, Section 3.

3. Supervised Learning, no generative model: Linear Regression
4. Supervised Learning, no generative model, classification: Logistic Regression
5. Supervised Learning, generative learning algorithms, classification: Gaussian Discriminant Analysis (GDA) a.k.a. MAP rule with Guassian likelihoods.
6. Supervised Learning, generative learning algorithms, classification: Spam Filter design (discrete-valued features)
7. Supervised Learning, generative learning algorithms, classification: Naive Bayes' assumption (assume independence of different features conditioned on class label): helps reduce number of parameters.
    - For GDA, this means the covariance matrix of the data is diagonal. If first apply PCA on the feature vectors before using GDA, then this assumption is automatically correct approximately (it is correct exactly in the limit of large m).
8. Supervised Learning, classification: Support Vector Machines (SVM)
    - Linear Classifier define: any classifier whose decision rule is of the form $\boldsymbol{w}^T\boldsymbol{x} + b > 0$ implies class 1 and ≤ 0 implies class -1.
    - Both of Logistic Regression and GDA (with equal covariance under all classes) are Linear Classifiers
    - SVM idea: find a $\boldsymbol{w}, b$ so that the margin is maximized for the data vectors that are the most likely to be mis-classified, i.e., idea is to maximize $\min_i y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b)$
    - SVM primal form optimization problem: it is a Quadratic Program (QP)
    - SVM dual form and why it is a good idea : can directly be used with Kernel SVMs
    - Kernel-ization idea and Kernel SVM.
9. PCA
    - r-SVD solution and its implications
    - Goal and what all it optimizes and why (proofs): proofs skipped this year.
    - Use for optimizing bias-variance tradeoff in Regression or in Classification problems: learn via homeworks
    - Picking the reduced dimension r: either use a percentage energy rule, or pick r to optimize Test-MSE (computed via cross-validation) for the final Regression or Classification problem being solved. Understand by doing HW 4: Linear Regression with PCA
10. Bias-Variance Tradeoff in the context of Linear Regression
    - Use prior on θ: entries of θ are arranged in decreasing order of magnitude or that θ is sparse or that θ is close to a θ_0 that is known
    - Use prior on training data features: model training data matrix $\boldsymbol{X}$ as being approximately low rank. This may not exactly hold but all n singular values will also not be equal almost all the time. Whenever this is the case – singular values are not all equal – one can use cross-validation to pick the best value of r.

11. Reading material used so far:

    - My notes below on these topics,
    - Homeworks 1a, 1b, 2a, 2b, 3, 4,
    - Andrew Ng's notes (labeled ML-cs229-) on Regression - linear and logistic, and on Generative Learning - contain more details and examples than mine
    - Andrew Ng's notes (labeled ML-cs229-) on SVMs
    - We have not used Ng's notes on PCA

18.7 Part 2: after March 25

Topics still to be covered

1. Unsupervised Learning: Clustering

    - Problem definition: given a dataset (set of features), partition it into k clusters (or classes) so that some "distance" within each class is as small as possible while "distance" between classes is as large as possible. k can be specified or may be unknown, one may have to pick k to optimize bias-variance tradeoff again.
    - k-means clustering
    - Gaussian Mixture Model (GMM): a commonly used generative model for clustering
    - EM algorithm for "learning" the GMM parameters: once GMM is learnt, clustering is done.
    - The General EM algorithm: extra learning. Will be skipped but notes are available in these Summary-Notes.

2. Deep Learning

    - Use Summary-Notes , will add Neural-nets file also into this file itself

3. Learning Theory and Bias-Variance Tradeoff

    - Use these SummaryNotes notes and use ML-cs229- notes

4. Extra topics from EE 527 (covered without proofs)

    - More on Least Squares estimation: particularly Recursive LS and Regularized LS
    - Bayesian (Generative) Learning more topics: Min Mean Squared Error (MMSE) estimation, Kalman Filtering, Hidden Markov Models (HMMs).

5. Homeworks 5, 6

19 Next few lectures

HW 5 (extra credit): 1. Re-do the "learning" part of HW 1 (linear regression) – for learning we are trying to estimate an nx1 vector θ using m training data samples, x^i, y^i. Suppose that the training data comes in sequentially; and/or suppose we want to keep updating our model as more training data comes in.

Consider the following setting: we initially have m_0 training samples, from which we can get an estimate $\hat{\theta}^{m_0}$ by regular LS.

Now suppose after this, we get one x^i, y^i pair at a time. This means the matrix X is growing in terms of number of rows and so is the vector $\boldsymbol{y}$. Use Recursive LS to get θ^i. Start with $\hat{\theta}^{m_0}$ and $\Pi = (X_0^T X_0)^{-1}$ where X_0 is the $m_0 xn$ matrix formed by the initial training samples.

2. In this homework itself or HW 2, re-visit the case where you do not have enough training samples – m is small. In this case use a prior with $\hat{\theta}_0 = 0$ and $\Pi = (1/\epsilon)I$, with $\epsilon = 0.00001$ or some small number.

Distill out some topics from my Estimation/Detection notes: MMSE estimation; Least Squares, Regularized LS, and Recursive LS; Kalman Filter; HMM. Teach the problem setting and algorithm; skip the proofs.

One/two lecture version of Sparse Recovery, Matrix Completion (Low-Rank Matrix Recovery), Robust PCA, Phase Retrieval

Homework 1
EE425X - Machine Learning: A signal processing persepective

In this homework you will be learning a Linear Regression model on two types of data sets. The problem here is as follows. You are given a set of m independent training data points, $y, \boldsymbol{x}$ that satisfy

$$y = \theta^T \boldsymbol{x} + e$$

and your goal is to estimate θ using training data that is either simulated or real. Denote the estimate as $\hat{\theta}$. Here y is a scalar and $\boldsymbol{x}$ is an n length vector.

1 Simulated (Synthetic) Data

1.1 How to generate the data

Generate your own data to simulate the linear regression model $y = \boldsymbol{\theta}^T \boldsymbol{x} + e$. Generate m such independent training data vectors. Also generate m_{test} independent data vectors for the testing step.

Let $\mathcal{N}(\mu, \boldsymbol{\Sigma})$ denote the Gaussian distribution with mean μ and covariance $\boldsymbol{\Sigma}$.

Do this as follows:

(a) Fix $\boldsymbol{\theta}$ once as specified below.

(b) Training data generate: For $i = 1$ to m, generate $\boldsymbol{x}^{(i)} \sim \mathcal{N}(0, I)$, $e^{(i)} \sim \mathcal{N}(0, \sigma_e^2)$ and $y^{(i)} = \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} + e^{(i)}$.

(c) Test data generate: In a different Loop, repeat (b) for $i = 1$ to m_{test}. Use this data in the testing step and not for training.

Generate the data for the following settings:

(a) Use $m = 30, 100, 1000$, $n = 5$, and set $e = 0$. Let $\boldsymbol{\theta} = [1, 4, 2, 10, 23]^T$.

(b) Pick $m = 30, 100, 1000$, $n = 5$, and set $\sigma_e^2 = 10^{-6}$. Let $\boldsymbol{\theta} = [1, 4, 2, 10, 23]^T$.

(c) Pick $m = 100, 100, 1000$, $n = 5$, and set $\sigma_e^2 = 10^{-4}$. Let $\boldsymbol{\theta} = [1, 4, 2, 10, 23]^T$.

(d) Repeat (b) with generating $\boldsymbol{x}^{(i)} \sim \mathcal{N}(\mu_x, I)$ with $\mu_x = [1, 1, 1, 1, 1]^T$. For this part, you will need to use a "mean value" in your model too, so see Sec 3.4 of Summary-Notes-2.pdf

(e) Repeat (b) generating $\boldsymbol{x}^{(i)} \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma}$ being diagonal. Specify in your report what diagonal entries you chose for $\boldsymbol{\Sigma}$ and what effect that had. Pick anything other than all equal entries.

Extra Credit: (f) Repeat (b) with generating $\boldsymbol{x}^{(i)} \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma}$ being a general covariance matrix. Specify in your report how you picked $\boldsymbol{\Sigma}$ and why.

Reporting: You are not reporting anything for this part.

1.2 Learning theta (Training)

Use the training data set here.

Use all three types of approaches for training, sample code for which is given in the Python-intro handout: Pseudo-Inverse, Solution of Normal Equations, and Gradient-Descent. Report the normalized error $\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2 / \|\boldsymbol{\theta}\|_2$ and the time taken in each case.

Reporting: Plot 1: Plot error versus m for all the 3 approaches in part (a) above in a A SINGLE FIGURE. Do the same for part (b) At least for part (a), all three should return an error value that is nearly zero.

Plot 2: Fix one approach for learning θ, let us say Pseudo-Inverse. Plot error versus m for parts (a), (b), (c) on A SINGLE FIGURE. This figure should help you understand how the error increases as we increase σ_e^2. Also, for a given σ_e^2, the error should decrease as m is increased.

Also comment on what you observe.

Other parts: Find the best way to report information from the other parts. It need not be a plot, it could be just a table or a bar plot. A good comparison for part (d) would be: how does the error change if you did not use the "mean" parameter θ_0 in your modeling versus if you did.

For part (e): you could take the figure from part (b), and add this plot to that figure to generate a new figure. See how the error from this part compares with that from the other parts.

Also, for this part: instead of the error norm, you can look at the error in the different components of the vector θ and see whether some components have larger error than others. Comment on how this is related to the "signal-to-noise ratio (SNR)", Σ_{ii}/σ_e^2, in the different components.

1.3 How to structure your code

Write your code so it is "general": read the entire homework to decide how general the code needs to be. For instance, do not hard-code in the value of m or n.

Your simulated data code should consist of the following parts

- Data Generation code: generate data as above.
- 3 pieces of code for learning θ using the 3 training approaches specified in Summary-Notes. So this provides 3 different estimates of the $n = 5$ length vector theta. Call these $\hat{\theta}$. This piece of code DOES NOT USE KNOWLEDGE OF the true θ or the true e. It only uses $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, i = 1, 2, \ldots, m$.
- Error computation: This code uses the true theta from the data generation code and estimates from the learning code and computes the normalized error.
- A Wrapper code that calls all the above three parts for different choices of $m, n, \sigma_e^2, \mu_x, \boldsymbol{\Sigma}$.

Reporting: No reporting needed for this part. This part just explains how to structure your code.

2 Real Data

In this problem you will be applying linear regression to real world data. Download the data from `https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise#`.

This contains $m = 1503$ training data points and there are $n = 5$ features. The last column of this data-set (6-th) column represents the output, y.

Here, you do not have access to the true θ so instead report the following error metric: $\sum_{i=1}^{m}(y^{(i)} - \hat{\boldsymbol{\theta}}^T\boldsymbol{x}^{(i)})^2/m$.

This is a large data set, so only implement Gradient Descent. Comment on the choice of `max-iter` and the learning-rate μ.

Extra Credit: Now, standardize the features (1-5 column) to ensure they have zero mean and unit variance, and repeat the experiment. One thing need to mention here is you are asked to standardize within each feature. One way to do that is calculate the mean and variance of each column and modify your data correspondingly. Also to have better result, you may want to use the intercept term in your algorithm. Report the same results.

Extra Credit: Implement (batch) Stochastic Gradient Descent and comment on how the batch-size affects convergence.

3 What to turn in?

Submit a short report that discusses all of the above questions, including results and analysis. Also submit your codes with clear documentation. Grading will be based on the quality of report and accuracy of implemented codes. I understand there are tons of good machine learning package you could find, like *scikit-learn* etc.. But forget them for today please, you could finish your homework only with help of *numpy* and necessary data importing package (like *pandas*).

Homework 1b: Linear Regression part 2.
EE425X - Machine Learning: A Signal Processing Perspective

Homework 1 focused on learning the parameter θ for linear regression. In this homework we will first understand how to use the learnt parameter to predict the output for a given query input. We will also understand bias-variance tradeoff and how to decide the model dimension when limited training data is available. This HW will rely heavily on the code from the previous homework.

Generate Data Code: Generate $m + m_{test}$ data points satisfying

$$y = \theta^T \boldsymbol{x} + e$$

with θ being ONE fixed n length vector for all of them. Use $n = 100$, $\theta = [100, -99, 98, -97...1]'$, $\sigma_e^2 = 0.01||\theta||_2^2$, $e \sim \mathcal{N}(0, \sigma_e^2)$, $\boldsymbol{x} \sim \mathcal{N}(0, I)$, and assume mutual independence of the different inputs and noise values (e).

1. Use code from Homework 1 (using any one approach is okay) to learn θ. Vary m and show a plot of both estimation error in θ,
$$||\theta - \hat{\theta}||_2^2 / ||\theta||^2$$
and a second plot of the "Monte Carlo estimate" of the prediction error on the test data (test data MSE).
$$\text{Normalized-Test-MSE} := \mathbb{E}[(y_{test} - \hat{y})^2]/\mathbb{E}[y_{test}^2], \text{ with } \hat{y} := \hat{\theta}^T \boldsymbol{x}_{test}$$
Monte Carlo estimate means: compute $(y_{test} - \hat{y})^2$ for m_{test} different input-output pairs and then average the result.
   (a) Vary m: use $m = 80$, $m = 100$, $m = 120$, $m = 400$. If your code is unable to return an estimate of θ, you can report the errors to be ∞ (and for the plot just use a large value say 100000 to replace ∞.
   (b) Repeat this experiment with $\sigma_e^2 = 0.1||\theta||_2^2$.
   Thus this part will produce four plots.
2. In this second part, suppose you have only $m = 80$ training data points satisfying $y = \theta^T \boldsymbol{x} + e$, with $n = 100$. *Notice n is the same as in the first part.* **I had a typo earlier which has now been fixed.**
   What you will have concluded from part 1 is that you cannot learn θ correctly in this case because m is even smaller than n.
   Let us assume you do not have the option to increase m. What can you do? All you can do is reduce n to a value $n_{small} \leq m$. Experiment with different values of n_{small} to come up with the best one. Do this experiment for two values of σ_e^2: $\sigma_e^2 = 0.01||\theta||_2^2$ and $\sigma_e^2 = 0.1||\theta||_2^2$.
   How to decide which entries of $\boldsymbol{x}$ to throw away? For now, just throw away the last $n - n_{small} + 1$ entries. So for $n_{small} = 1$, let $\boldsymbol{x}_{small}$ be just the first entry, and so on. So for $n_{small} = 30$ for example, $\boldsymbol{x}_{small}$ will be the first 30 entries of $\boldsymbol{x}$. There are many other better ways which we will learn about later in the course.
   Start with $n_{small} = 1$ and keep increasing its value and each time compute Normalized-Test-MSE by learning a value of θ first (using $m = 80$ of course). Obtain a plot. Use the plot and what you learn in class to decide what value of n_{small} is best.
3. Interpret your results based on the Bias-Variance tradeoff discussion. See Section 11 of Summary-Notes and what will be taught in the next few classes.

Homework 2
EE425X - Machine Learning: A signal processing persepective
Logistic Regression and Gaussian Discriminant Analysis

In this homework we are going to apply Logistic Regression (LR) and Gaussian Discriminant Analysis (GDA) for solving a two-class classification problem. The goal will be to implement both correctly and figure out which one is better.

To do this, you will first "learn" the parameters for each case using the training data (as discussed in class and available in the handouts). Then, you will apply it to test data and evaluate the performance as explained below. The only change from the handout is that, for GDA, **you need to assume that the covariance matrix $\mathbf{\Sigma}$ is diagonal.**

1 Synthetic Data Generation

Generate your own training data first. To do this, we use the GDA model because that is the only one which provides a generative model.

- Generating Training data: Since we want to implement a two-class classification problem, let the class labels, $y^{(i)}$ take two possible values 0 or 1 (for $i = 1, \cdots, m$, i.e., we have m training samples). These are generated independently according to a Bernoulli model with probability ϕ. Next, conditioned on $y^{(i)}$, the features $\boldsymbol{x}^{(i)} \in \mathbb{R}^{n\times 1}$ are generated independently from a Gaussian distribution with mean $\mu_{y^{(i)}}$ and covariance matrix $\mathbf{\Sigma}$. In other words, while generating $x^{(i)}$, use the same covariance matrix $\mathbf{\Sigma}$ for both classes, but pick two different μ's: μ_0 as the n-dimensional mean vector for data from class 0 and μ_1 as the n-dimensional mean vector for data from class 1. Do this for all $i = 1, 2, \cdots, m$.
- Generating Test data: Do the same as above, but now instead generate $m_{test} = m/5$ samples.
- For the synthetic data part of this homework, use $n = 100$ and $m = 20$.

2 Learning parameters using training data; and then testing the method on test data

2.1 Training

- Write code to estimate the parameters for Logistic Regression and for GDA. For how to do it, please refer to the class handouts. GDA was covered recently in the Generative Learning Algorithms handout. LR is covered in the first handout (Supervised Learning).
- For LR, you need to write Gradient Descent code to estimate θ. Refer to the handouts, and the optimization_intro notebook on Canvas for more details.

- For GDA, proceed as follows. The ONLY CHANGE from the handout is that we assume that $\mathbf{\Sigma}$ is DIAGONAL and thus use the following formulas:

$$\phi = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=1)$$
$$\mu_0 = \frac{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=0)\boldsymbol{x}^{(i)}}{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=0)}$$
$$\mu_1 = \frac{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=1)\boldsymbol{x}^{(i)}}{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=1)}$$
$$(\mathbf{\Sigma})_{k,k} = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}^{(i)}-\mu_{y^{(i)}})_k^2,\ k=1,2,\ldots,n$$

  while setting all non-diagonal entries of $\mathbf{\Sigma}$ to be zero. Here, $\mathbf{1}(w=c)$ is the indicator function that evaluates to 1 when $w=c$ and 0 otherwise.

2.2 Testing

Now, for both LR and GDA, you have estimated the parameters, namely, θ for Logistic Regression, and, $\phi, \mu_0, \mu_1, \mathbf{\Sigma}$ for Gaussian Discriminant Analysis. Once you have these parameters, evaluate how "good" each method is by computing the testing accuracy.

- For LR, for each (test) input query $\boldsymbol{x}$, compute the output $\hat{y}_{LR}(\boldsymbol{x})$ as

$$\hat{y}_{LR}(\boldsymbol{x}) = 1 \text{ if } h_{\hat{\theta}}(\boldsymbol{x}) > 1 - h_{\hat{\theta}}(\boldsymbol{x}),$$

  and $\hat{y}=0$ otherwise. In the above equation, $h_{\hat{\theta}}(\boldsymbol{x}) = \sigma(\hat{\theta}^T\boldsymbol{x})$. And again, recall that $\hat{\theta}$ is what you estimated using the training data and implementing Gradient Descent.

- For GDA, we use Bayes rule for classification. For each input query $\boldsymbol{x}$, compute the output $\hat{y}_{GDA}(\boldsymbol{x})$ as

$$\hat{y}_{GDA}(\boldsymbol{x}) = \arg\max_{l\in\{0,1\}} \mathcal{N}(\boldsymbol{x};\mu_l,\mathbf{\Sigma})\phi^l(1-\phi)^{1-l}$$

- Evaluate error: let us denote the test data as D_{test}. Report error of each method as

$$\text{error} = \frac{1}{|D_{\text{test}}|}\sum_{(\boldsymbol{x},y)\in D_{\text{test}}} |y-\hat{y}(\boldsymbol{x})|$$

  where $\hat{y}(\boldsymbol{x})$ is the output of the classifier for input $\boldsymbol{x}$. Also, $|D_{\text{test}}| = m_{\text{test}}$ is number of testing samples. YOU NEED TO DO THIS FOR BOTH METHODS.

3 Real Data

Next use the MNIST dataset to evaluate both approaches on real data. MNIST is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. The entire dataset can be downloaded from here but in this problem we only use samples corresponding to two digits 0 and 9.

Use the code written in the previous part to classify two digits 0 and 9 in MNIST by using Logistic Regression and Gaussian Discriminant methods. You should have written code for part 2 so you need not have to rewrite anything, except change what you provide as training and test data. This is what we want

to learn in this course: use simulated (synthetic) data to write and test code; make sure everything works as expected, then use the same code on real data.

Please report the final classification accuracy and discuss how the obtained accuracy for the real data differences from the synthetic data.

4 What to turn in?

Submit a short report that discusses all of the above questions. Also submit your codes with clear documentation. Grading will be based on the quality of report and accuracy of implemented codes.

Homework 3
EE425X - Machine Learning: A signal processing perspective

In this homework we will use Support-Vector Machine (SVM) to classify MNIST dataset. Same as the previous homework, we only consider two classes of data in MNIST, classes associated with digits 0 and 9.

1 SVM for MNIST

Basically in SVM we need to learn function,

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b,$$

so that $g(\boldsymbol{x}) > 0$ means image $\boldsymbol{x}$ belongs to class 9 and vice versa. Remember that in MNIST image $\boldsymbol{x}$ is a 28×28 matrix and $\boldsymbol{w}$ is the same size of $\boldsymbol{x}$.

- Use training data to estimate $\boldsymbol{w}$ and b in SVM method. More details about updating $\boldsymbol{w}$ and b can be found in your class notes.
- Use $\boldsymbol{w}$ and b obtained from the previous step to classify test data.
- Use the test accuracy metric from the previous homework to evaluate performance

2 Want to learn more!

You don't have to do this problem and this problem doesn't have extra points. One can get more accurate classifier by using Kernels.

- Repeat the previous problem for polynomial kernel of size 5, that is $g(\boldsymbol{x}) = (\boldsymbol{w}^T \boldsymbol{x})^5 + b$.
- Repeat the previous problem for Gaussian kernel, that is $g(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{w}\|^2 / 2\sigma^2\right) + b$.

3 What to turn in?

Submit a short report that discusses all of the above questions. Also submit your codes with clear documentation. Grading will be based on the quality of report and accuracy of implemented codes.

Homework 4: Using PCA for Model Selection in Linear Regression
EE425X - Machine Learning: A Signal Processing Perspective

Homework 1 focused on learning the parameter θ for linear regression. In this homework how to use PCA for model selection; this is especially relevant when limited training data is available.

1 Generate Data

Use $n = 100$, $m_{tot} = 80$.

Generate m data points satisfying

$$y = \theta^T \boldsymbol{x} + \epsilon$$

with θ being a fixed n length vector for all of them.

- Can generate θ any way you want. Let us fix it at $\theta = [100 : -0.5 : 50.5]'$;
- Generate n-length vectors $\boldsymbol{x}^{(i)} \sim \mathcal{N}(\mu_x, \Sigma_x)$ with $\mu_x = [5 : 0.1 : 14.9]'$ and $\Sigma_x = tmp * evals * tmp'$ with $tmp = randn(n, n)$ and $evals = diag([100 * randn(round(n/8), 1).^2; randn(n - round(n/8), 1).^2])$ is a diagonal matrix. Here $.^2$ means each entry of the vector is squared (MATLAB notation). Arrange these as rows of an $m_{tot} \times n$ matrix $\boldsymbol{X}_{tot}$.

  Repeat the experiment with also trying $evals = diag([100 * randn(n, 1).^2])$.

  Note: to generate $\boldsymbol{x} \sim \mathcal{N}(\mu_x, \Sigma_x)$, you do $\boldsymbol{x} = \mu_x + (tmp * sqrt(evals) * tmp') * randn(n, 1)$.
- For any experiment, we can either use leave-one-out-cross-validation (most efficient use of the data, but it is time-consuming) or since this is just a HW to demonstrate a few ideas, we can keep things simpler. Split $m_{tot} = 80$ into two parts, $m = 50$ and $m_{test} = m_{tot} - m = 30$. Use the first m for training and the next m_{test} for computing the test-MSE.

  So we have $\boldsymbol{X} = \boldsymbol{X}_{tot}(1 : m, 1 : n)$ and $\boldsymbol{X}_{test} = \boldsymbol{X}_{tot}(m + 1 : m_{tot}, 1 : n)$. Similarly $y = y_{tot}(1 : m, 1)$ and $y_{test} = y_{tot}(m + 1 : m_{tot}, 1)$.

2 Write the algorithm to pick the best model

Let us assume you do not have the option to increase m. What can you do? All you can do is reduce n to a value $r \leq m$. This can be done in various ways depending on what "prior knowledge" is available about either θ or the training data points $\boldsymbol{x}^i$'s.

1. Methods that use priors on θ: suppose you knew that entries of θ were in decreasing order of magnitude from 1 to n. Then you would try to use the approach used in HW1b. Retain the first r entries of each data points $\boldsymbol{x}^{(i)}$; then learn an r-length estimate $\hat{\theta}_r$; for each of the test data points: use this on the first r entries of test data vector to get a prediction $\hat{\boldsymbol{y}} = \hat{\theta}_r^T \boldsymbol{x}_{test}(1 : r)$, compute the squared error $(y - \hat{\boldsymbol{y}})^2$ and compute an average of this quantity over all test data. This is an approximation to $Test - MSE = \mathbb{E}[(y - \hat{\theta}_r^T \boldsymbol{x}_{test}(1 : r))^2]$.

   Repeat above process for different values of r and store the Test-MSE value for each r. Pick the best r.

2. Methods that use priors on θ: suppose you knew that θ is exactly or approximately sparse. The example given in the first item above is a simpler special case of this assumption. We will not explore this in the current HW.

3. Methods that use a prior on the $m \times n$ training data matrix $\boldsymbol{X}$: suppose that we know that $\boldsymbol{X}$ is approx low rank, meaning that its rank is smaller than $\min(m, n)$. We will proceed as follows.
   - Repeat the following for each value of $r = 1, 2, \ldots, n$ (or can just go till m because we know error will blow up for $r > m$).
     (a) Compute $\hat{\mu} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
     (b) Compute $\boldsymbol{Z} = \boldsymbol{X} - \mathbf{1}\hat{\mu}^T$
     (c) Compute SVD of $\boldsymbol{Z}$, $\boldsymbol{Z} = \boldsymbol{U}_{full} \boldsymbol{S}_{full} \boldsymbol{V}_{full}^T$.
     (d) Set $\boldsymbol{V} = \boldsymbol{V}_{full}(:, 1:r)$: this is the principal subspace often also called "PCA space".
     (e) Compute $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{V}$: these are the projections into the principal subspace.
     (f) Include a "mean" term in the regression model, c. To do this, let $\tilde{\boldsymbol{B}} := [\mathbf{1}, \boldsymbol{B}]$. Here $\mathbf{1}$ is a vector of ones.
     (g) Compute $\hat{\tilde{\theta}}_V = \tilde{\boldsymbol{B}}^{\dagger} \boldsymbol{y}$ where $M^{\dagger} = (M^T M)^{-1} M^T$.
     (h) $\hat{c} = \hat{\tilde{\theta}}_V(1, 1)$ and $\hat{\theta}_V = \hat{\tilde{\theta}}_V(2 : r+1, 1)$
     (i) Compute $\hat{\boldsymbol{y}}_{test} = \boldsymbol{X}_{test}(\boldsymbol{V}\theta_V) + \hat{c}$
     (j) Compute Normalized-Test-MSE(r) $= ||\boldsymbol{y}_{test} - \hat{\boldsymbol{y}}_{test}||_2^2 / ||\boldsymbol{y}_{test}||_2^2$
   - Plot the Normalized-Test-MSE and select the best value of r.

   Note: because everything is linear and because we are using a "mean" term c, in this experiment, it does not matter whether or not we first subtract the mean before doing PCA. It is possible that mean subtraction is important when using PCA as a pre-processing step for classification though

3 Real Data

In this problem you will be applying linear regression to real world data.

1. Download the data from `https://archive.ics.uci.edu/ml/datasets/BlogFeedback`.
2. This contains various files, but for this homework, *only use the first two files for training*, i.e., use *both* `blogData_test-2012.02.01.00_00.csv` and `blogData_test-2012.02.02.00_00.csv`.
3. The total number of rows will be $m = 248$. The number of features, $n = 280$. *You may need to transpose this matrix to get the matrix $\boldsymbol{X}_{train}$ in the standard form that we assume – remember we are writing each feature (or input) vector as a row.*
4. The last column of each of these files is the output, y.
5. Now consider the `blogData_test-2012.02.03.00_00.csv` as the test data (only the first 280 columns), and report the Normalized-Test-MSE as explained above.

4 What to turn in?

Submit a short report that discusses all of the above questions. Also submit your codes with clear documentation. Grading will be based on the quality of report and accuracy of implemented codes.

Homework 5: Using PCA before Classification
EE425X - Machine Learning: A Signal Processing Perspective

Homework 2 focused on classification using logistic regression and GDA. In this homework I would like you to do the same two tasks, but first use PCA to reduce the dimension of the data. This is a common practice in ML known as pre-processing the data.

As in HW 4, use cross-validation to pick the best value of r. You can use a simple cross validation as done in HW 4: split the available m_{tot} data points into $m_{tot} = m + m_{test}$. Use m points for training and the rest m_{test} for testing. Compute Test-Error as explained in HW 2. Do this for each value of r. Pick the best r.

Simulated data: generate data as suggested in HW 2 BUT with the following difference: use the following covariance matrix Σ: $\Sigma = tmp * evals * tmp'$ with $tmp = randn(n, n)$ and $evals = diag([100 * randn(round(n/8), 1).^2; randn(n - round(n/8), 1).^2])$ is a diagonal matrix of the eigenvalues of the covariance. Here $.^2$ means each entry of the vector is squared (MATLAB notation). Repeat the experiment with $evals = diag([100 * randn(n, 1).^2])$.

When you are doing the above, you need to be careful that the class mean for class zero, μ_0, is sufficient "far" from the class mean for class one, μ_1. To be precise, $(\mu_0 - \mu_1)^T \Sigma^{-1} (\mu_0 - \mu_1)$ should be "large". This is not easy to simulate. Instead if you just make sure $||\mu_0 - \mu_1||_2^2$ is large compared to $trace(Sigma)$ (this is also equal to $sum(eigenvalues(Sigma)) = sum(evals)$), this should suffice for you to get a good classification accuracy. In particular, roughly $||\mu 0 - \mu 1||_2^2 > 9 trace(\Sigma)$ should work (3-Sigma rule).

Comment on which model (logistic regression vs GDA) works better. Also compare with the results obtained in HW2 (do not perform PCA).

Real dataset: (a) use the MNIST dataset (only class 0 and 9) as is; (b) use the MNIST dataset but use a small value of m.

Homework 6: Clustering, and Clustering wtih PCA as pre-processing step
EE425X - Machine Learning: A Signal Processing Perspective

This homework focuses on clustering which means we are given a set of m feature vectors (each vector is n length). We would like to cluster these features into k clusters. For this HW (except for the extra credit part) let us assume k is known. Suppose $k = 3$.

To generate the data, we will generate it exactly as in HW 5 (not HW 2) using the GDA model, but now we have $k = 3$ classes (instead of 2 classes earlier). So instead of a single ϕ, we now have $\phi_j, j = 1, 2, 3$ such that $\sum_j \phi_j = 1$. We can set them as $\phi_1 = 0.3, \phi_2 = 0.4, \phi_3 = 0.3$ for example.

Generate class label $y^{(i)}$ as follows: $p(y^{(i)} = j) = \phi_j, j = 1, 2, 3$. Given $y^{(i)} = j$, generate $\boldsymbol{x}^{(i)}$ from $\mathcal{N}(\mu_j, \Sigma)$, with
$\Sigma = tmp * evals * tmp'$ with $tmp = randn(n, n)$ and $evals = diag([100 * randn(round(n/8), 1).^2; randn(n - round(n/8), 1).^2])$ is a diagonal matrix. Here $.^2$ means each entry of the vector is squared (MATLAB notation). Repeat the experiment with also trying $evals = diag([100 * randn(n, 1).^2])$.

When using the data in the clustering algorithm, you ONLY use $\{\boldsymbol{x}^{(i)}, i = 1, 2, \ldots, m\}$ and DO NOT USE the class labels $y^{(i)}$.

Clustering: suppose $n = 50$ and $m = 200$. Reduce the value of m in the PCA part if you do not see anything interesting with this value of m. Output of a clustering algorithm is

- the estimated class labels $\hat{y}^{(i)}, i = 1, 2, \ldots, m$
- estimated $\hat{\mu}_j, j = 1, 2, \ldots, k$.
- when using EM algo, it also outputs the covariance matrix estimates; when using k-means, it does not.

The clustering cost function is

$$J(\boldsymbol{x}^{(i)}) := \frac{1}{m} \sum_{i=1}^{m} ||\boldsymbol{x}^{(i)} - \hat{\mu}_{\hat{y}^{(i)}}||_2^2.$$

Do the following

1. implement k-means clustering (with multiple random initializations - pick best one as the one with smallest cost).
2. implement k-means clustering with PCA as a pre-processing step. As in the previous two homeworks, loop over r to decide the best value of r.
3. Real dataset: use the MNIST dataset with any $k = 3$ digits. Decide a reasonable value of m, need not use all the data points.

Extra Credit:

1. Extra Credit 1: implement EM algorithm as well
2. Extra Credit 2: assume k is not known; automatically decide k by computing the clustering cost function.

Project: combine ideas of 2-4 homeworks, and work on them in more detail. First report due April 22 for comments from me or Praneeth. Final project due: last day of finals' week.

Groups: groups of at most 3. Groups of 2 required.

-- Develop your own approach to write/combine code. In the report: write pseudo-code to explain exactly what you coded. What you say in the report should match up with what you coded in.

-- Derive your own conclusions - first based on simulated data (figure out the correct way to generate it and also to explain it in your report) - then based on real data. Report: Explain how to generate the simulated data and why this way.

-- Implement on one more dataset other than the ones we have provided in the HWs. Explain the pre-processing steps to use the dataset first, explain what you observe on real data and why. Set up your own reasonable error metrics or use existing ones. In Report: Explain the dataset, explain what you observe and why.

-- **Basically the report should tell me what all you did, how you coded it in (pseudo-code, not code attached), and what you observed, first for simulated, then for real data.**

Examples:

- Various pre-processing steps for Linear Regression : use of PCA versus use of sparsity on theta versus use of both (first do PCA --pick r as done in HW 4 -- then, on this reduced dimensional data, try to fit a sparse θ). So compare: (i) just PCA, (ii) just sparse prior on θ, (iii) combination of both.

- PCA as pre-processing step for linear regression, logistic regression and GDA (HW 4, 5)

- Various classification approaches: compare and contrast (HW 2, 3)

- PCA for classification and clustering (HW 5, 6)

- Welcome to also take any one HW and extend it beyond the existing HWs.

Deadlines:

- April 10: submit an "abstract" (< 1 page summary of what you will do)

- April 29: draft version of the project (if you want comments from me): strongly encouraged.

- May 5: Project due. Submit: (1) a PDF report, (2) code

Strongly encourage you to use LateX -- free, best quality reports, easy to use once you figure out the basics.

https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes (Links to an external site.)