ML - Master Notes

June 20, 2021

1 Elements of ML

1.1 Key Terms

- inference, prediction, classification, regression
- training set, test set, generalisation
- supervised learning, unsupervised learning
- clustering
- parametric model, non-parametric model
- mathematical optimization
- sum of squared errors: $E(w) = \frac{1}{2} \sum [y(x_n, w) - t_n]^2$ — $w* = argmin_w E(w)$
- closed form solutions/ analytic solutions, iterative algorithms
- over fitting, under fitting
- root mean squared error: $E_{RMS} := \sqrt{2E(w*)/N} \implies$ compares different data set sizes equally
- Regularization: accounting for the desire to not over-fit by including a penalty term in the error function. For example: $E_R(w) = E(w) + \frac{\lambda}{2} \sum_{i=0}^{n} w_i^2$, where λ is a regularization parameter governing the importance of the model complexity penalty
- validation set, k-fold cross validation
- leave one out cross validation: k-fold cross validation where k = n, the number of data points
- K-nearest neighbours
- uncertainty, frequentist, bayesian
- maximum likelihood principle
- bootstrap: use MLE to predict parameters for a model to fit multiple alternative data sets, and measure the variance of the parameters over the distribution of data sets For example, to bootstrap we may select a data sets from D by taking N data points with replacement

1.2 Regularisation

The key idea here is to achieve the perfect balance between bias (over-fitting) and variance (under-fitting). There are many ways of achieving this. For instance:

- setting **M**, the number of hidden units, can limit the complexity of the function which can be modelled.
- **Weight Decay**: start with a large value of **M** and add a regularisation parameter to the error function.
- **Early Stopping**: stop at the point of lowest error for the validation set, not the train set.

2 Regression

Definition of regression: to predict the continuous target value of one or more variables

2.1 Linear Regression

- Linear regression models only need to be linear with respect to the adjustable parameters.
- They do not need to be linear with respect to the input variables.
- Therefore, polynomial regression is also linear regression (e.g. $f(x) = \alpha x^2$) but multi-layer neural networks are not.
- For more advanced linear regression, we can take a linear combination of a number of non linear functions of the input variables.
- Example of a linear model: $F(X, w_1, w_2) = w_1 * \sin x + w_2 * x^2$
- Example of a non linear model: $F(X, w_1) = w_1^2 * x$
- The non-linear functions we use on the input variables are called basis functions
- Single layer neural networks which sum the weighted inputs are linear models!

2.2 Error function

Choice of *error function* is linked to maximum likelihood principle.

Let us do some analysis where the key assumption is that noise is gaussian:
Let our training set be the set of (x_n, t_n) pairs where x is an input vector and t is the output value.

We assume there is a true deterministic function giving rise to t: $t = t(x, w) + \epsilon$, where ϵ is gaussian noise with zero mean and variance σ^2

So we can write

$$p(t|\mu, w, \sigma^2) = N(t|y(x, w), \sigma^2)$$

$$p(t_1, ..., t_N|x_1, ...x_N) = \prod_{n=1}^{N} N(t_n|y(x_n, w), \sigma^2)$$

Now we can define the log likelyhood as:

$$L(\theta) = log \prod_{n=1}^{N} N(t_n|y(x_n, w), \sigma^2)$$

$$= \sum_{n=1}^{N} log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\mu)}{\sigma}^2}$$

$$= N \log \frac{1}{\sqrt{2\pi}} + \sum_{n=1}^{N} \log e^{-\frac{1}{2}\frac{(x-\mu)}{\sigma}^2}$$

$$= N \log \frac{1}{\sqrt{2\pi}} + \sum_{n=1}^{N} -\frac{1}{2}\frac{(x-\mu)}{\sigma}^2$$

2.3 Gradient Descent

The gradient descent algorithm is described below, given a training set and basis functions ϕ

First we need to define the cost function, $J(\theta)$:

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2} ||h_\theta(x^n) - y^n||^2 + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{l=1}^{s_l} \sum_{j=1}^{s_l+1} W_{ji}^{s_l+1^2}$$

1. Initialize random weights $\boldsymbol{w}$, set regularization parameter λ, stopping condition ϵ, and training rate η
2. While the difference between weights from the previous step is less than ϵ:
    3. Randomly shuffle the training data rows
    4. For each training example input and output pair $(\boldsymbol{x}_n, y_n)$:
        5. Calculate the gradient vector using $\frac{d\boldsymbol{e}_{L2}}{d\boldsymbol{w}} = -(y_n - \boldsymbol{w} * \phi(\boldsymbol{x_n}))\phi(\boldsymbol{x_n}) + \lambda \boldsymbol{w}$
        6. Perform weight update: $\boldsymbol{w} = \boldsymbol{w} - \eta \frac{d\boldsymbol{e}_{L2}}{d\boldsymbol{w}}$

3 Classification

[1]

In classification, you can either model a decision boundary itself (e.g. a set of hyper-planes) - this would be called a **discriminative model**, or a probabilistic generative model (e.g. a bayesian classifier) which models the probability that

3.1 Discriminative Models

Definition: Non probabilistic models which directly assign any input to one class.

Perceptron

The perceptron for *two classes* is a generalised linear model with a step function deciding which class to apply to, i.e.:

$\hat{y} = f(\mathbf{w} * \phi(x) + b)$, where f is 1 for values greater than 0, else -1.

Since it is a non differentiable function, we must use the "perceptron update" function instead of back prop.

3.2 Probabilistic Generative Models

Definition: Infer the class posterior from the class prior and the class conditional, which we derive from the data (e.g. Naive Bayes).

Benefits of generative models: by modelling the underlying distributions, we can generate synthetic data (e.g. by plugging the generated parameters into a probability model and using RNG).

Detriments of generative models: often they have more parameters than discriminative models.

Mathematically: $P(C_k|x) = P(C_k) * P(x|C_k)$

3.3 Probabilistic Discriminative Models

Definition: Probabilistic model which assigns a probability per class input pair.

4 Latent Variable Models

4.1 K-Means Clustering

- K-means is sensitive to the initial clusters - no guarantee it will converge to the same clusters each time
- K means is a form of clustering, which is a form of unsupervised learning

Algorithm 1 K-Means Clustering

procedure K-MEANS($\mathbf{X}, K$) ▷ $\mathbf{X}$ are points $\{x_n\}, n \in 1..N, K clusters$
  $C \leftarrow initClusters(K)$ ▷ C is a set of clusters $\{c_i\}, i \in 1..K$
  while $C' \neq C$ **do** ▷ While the clusters are not stable
    $C \leftarrow C'$
    for $x \in \mathbf{X}$ **do**
      cluster(x) $\leftarrow argmin_{i \in 1..C}(\text{dist}(c_i, x))$
    for $c \in C'$ **do**
      c $\leftarrow$ average([x $\forall$ cluster(x) == c])
  return C ▷ Return the clusters

4.2 Gaussian Mixture Models

Here we consider more concretely how a hypothetical sample space **which could be well clustered** might arise:

4.3 The environment

(k, x) pairs are generated by performing the following steps:

- sample k from $1..K$ with $\phi_k = P(K = k) => \sum_{k=1}^{K} \phi_k = 1$
- each k corresponds to one of a collections of (possibly multi-variate) Gaussian distributions, with parameters $\mu_{\mathbf{k}}$ and $\sum_{\mathbf{k}}$
- to select a data point, we sample from the distribution $\mathcal{N}(\mu_k, \sum_k)$
- therefore we have a set of data points, however only the oracle knows the labels - referred to as **latent** variables. We only see the data points. It is our job to infer the possible labels through machine learning
- Therefore, our job is to learn both the labels for each data point, and some model parameters to model the underlying distributions of those labels: $\theta := (\phi, \mu_{\mathbf{1}}, \sum_{\mathbf{1}}, ..., \mu_{\mathbf{k}}, \sum_{\mathbf{k}})$
- note: this seems reminiscent of a hidden Markov model with continuous observed states.

4.4 Likelihood

The probability of a given label and data point pair is:
$P(k, x_n) = P(K = k)P(x_n|K = k) = \phi_k \mathcal{N}(\mathbf{x_n}|\mu_\mathbf{k}, \sum_\mathbf{k})$

Since we do not know k, we sum (marginalise) over the unknown:
$P(\mathbf{x}_n) = \sum_{k=1}^{K} \phi_k \mathcal{N}(\mathbf{x_n}|\mu_\mathbf{k}, \sum_\mathbf{k})$

Log Likelihood of the entire model θ:
$\mathcal{L}(\theta) = \sum_{n=1}^{N} \ln p(x_n)$

5 Hard EM for Document Clustering

5.1 Notation

Here we describe the notation used (from Alexandria)

- There are N documents $\{\mathbf{d}_1, ..., \mathbf{d}_N\}$, each having a cluster assignment vector (the latent variable) $\{\mathbf{z}_1, ..., \mathbf{z}_n\}$, which is a sparse matrix containing 1 for the entry corresponding to the cluster, and 0 for all others.
- $P(k) = \phi_k$, the probability of a document being of cluster k. There are K clusters
- set of words in a dictionary $w \in \mathcal{A}$
- word proportion vectors for each cluster μ_k, where $\sum_{w \in \mathcal{A}} \mu_{k,w} = 1$
- counts of word w in document d: $c(w, d)$
- The mixing components ($\phi_k = \frac{N_k}{N}$) are the total documents with a cluster over the total documents
- The word proportion parameters in each vector:

$$\mu_{k,w} = \frac{sum_{n=1}^{N} z_{n,k} c(w, d_n)}{\sum w' \in \mathcal{A} \sum_{n=1}^{N} z_{n,k} c(w', d_n)}$$

  , Which is the sum of all words in a cluster over the sum of all words in all documents of the cluster.
- $\theta := (\phi, \mu_\mathbf{1}, ..., \mu_\mathbf{k})$
- $\gamma(z_{nk}) := p(z_{nk} = 1|\mathbf{d}_n, \theta)$ is the probability that a document n is of cluster k

$$P(z_{n,k}) = \frac{\phi_k \prod_{w \in \mathcal{A}} \mu_{k,w}^{c(w,d_n)}}{\sum_{k=1}^{K} (\phi_k \prod_{w \in \mathcal{A}} \mu_{k,w}^{c(w,d_n)})}$$

Algorithm 2 Soft EM For Document Clustering

procedure SoftEmDoc($\mathcal{D} := d_1, ..., d_n, K$) ▷ $\mathcal{D}$ contains N documents, K is the number of clusters
  $\theta^{new} \leftarrow initTheta()$ ▷ $\theta := \{\phi, \mu_{\mathbf{1}}, ..., \mu_{\mathbf{K}}\}$
  do
    $\theta^{old} \leftarrow \theta^{new}$
    for $n \in N$ **do**
      for $k \in K$ **do**
        $\gamma(z_{n,k}) = P(z_{n,k} = 1|\mathbf{d}_n, \theta^{\mathbf{old}})$ ▷ Expectation (E) Step

$$\phi^{\mathbf{new}} = \frac{\sum_{n=1}^{N} \gamma(z_{n,k})}{N}$$

$$\mu_{\mathbf{k},\mathbf{w}} = \frac{\sum_{n=1}^{N} \gamma(z_{n,k}) c(w, d_n)}{\sum_{w' \in \mathcal{A}} \sum_{n=1}^{N} \gamma(z_{n,k}) c(w' d_n)}$$

▷ Maximisation (M) Step

  while $diff(\theta_{\mathbf{old}}, \theta^{\mathbf{new}}) < \epsilon$
  return θ^{new}

Algorithm 3 Hard EM For Document Clustering

procedure SoftEmDoc($\mathcal{D} := d_1, ..., d_n, K$) ▷ $\mathcal{D}$ contains N documents, K is the number of clusters
  $\theta^{new} \leftarrow initTheta()$ ▷ $\theta := \{\phi, \mu_{\mathbf{1}}, ..., \mu_{\mathbf{K}}\}$
  do
    $\theta^{old} \leftarrow \theta^{new}$
    for $n \in N$ **do**
      for $k \in K$ **do**
        $\gamma(z_{n,k}) = P(z_{n,k} = 1|\mathbf{d}_n, \theta^{\mathbf{old}})$ ▷ Expectation (E) Step

$$\phi^{\mathbf{new}} = \frac{\sum_{n=1}^{N} \gamma(z_{n,k})}{N}$$

$$\mu_{\mathbf{k},\mathbf{w}} = \frac{\sum_{n=1}^{N} \gamma(z_{n,k}) c(w, d_n)}{\sum_{w' \in \mathcal{A}} \sum_{n=1}^{N} \gamma(z_{n,k}) c(w' d_n)}$$

▷ Maximisation (M) Step

  while $diff(\theta_{\mathbf{old}}, \theta^{\mathbf{new}}) < \epsilon$
  return θ^{new}

6 Neural Networks

6.1 Formulas to remember

- Sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\mathrm{d}\sigma}{\mathrm{d}z} = \sigma(z)(1-(\sigma(z)))$$

- Tanh function:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\frac{\mathrm{d}tanh(z)}{\mathrm{d}z} = 1 - (tanh(z))^2$$

Normal Distribution Formula:

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

- Neuron:

$$H_{W,b} := f(W^T x) = f(\sum_i W_i x_i + b)$$

Textbook Terminology

- nl = number of layers
- layer L = L_l where L_1 is the input layer and layer L_{nl} is the output layer
- a_i^l = activation after f for neuron i at layer l. $a_i^1 = x_i$ and a_i^{nl} is the final output to be compared to the training data.
- z_i^l = weighted sum of inputs to neuron i at layer l.
- W_{ij}^l = Weight leading from neuron j in layer l to neuron i in layer l + 1.
- b_i^l = Bias term leading towards neuron i in layer l + 1.

6.2 Activation and Error Functions

They key for all of the below examples is to understand what probability distribution the target function is modelled under, and therefore how maximising the joint probability could lead to a maximum likelyhood error function.

Regression

Activation Function: In a regression problem, assuming we aren't making some claim about the range of our true output values, we will need the final output to be able to model any real valued number (or set therefore, but we focus on the single output case here). Therefore, while the hidden layers may have non-linear activation functions, the final layer must be linear (e.g. just the sum of previous layers.

Distribution:

Assume a Gaussian model:

$$P(y|x) = \prod_{n \in 1..N} \mathcal{N}(x_n W, \sigma)$$

After taking the negative log likelyhood and doing some math, we get the sum of squared errors:

Error Function:

$$J(\theta) = \frac{1}{2} \sum_{n \in N} (X_n W + b - y)^2$$

Single Class Classification

Activation Function

We assume need to output a probability of being in the singleton class, so the **sigmoid** function will work.

Distribution:

We must assume a binomial distribution, with probability $\prod_{n \in N} h_n^{y_n} (1 - h_n^{y_n})^{1-y_n}$

Error Function

When we take the negative log of this, we end up getting **Cross Entropy Loss**:

$$J(\sigma) = -\sum_{n \in N} y^n log(h_n) - (1 - y_n) log(1 - h_n)$$

Multi Class Classification

Activation Function

We will use **softmax** activation function: softmax transforms any real valued vector into a probability distribution with the following formula:

For each element i of vector v:

$$softmax(v_i) = \frac{e^{v_i}}{\sum_{v_n \in v} e^{v_n}}$$

Distribution:
We also binomial distribution, with probability $\prod_{n \in N} \prod_{k \in Classes} h_n^{y_{nk}} (1 - h_n^{y_{nk}})^{1 - y_{nk}}$

Error Function
When we take the negative log of this, we end up getting **Cross Entropy Loss**:

$$J(\sigma) = -\sum_{n \in N} \sum_{k \in Classes} y^{nk} log(h_n) - (1 - y_{nk}) log(1 - h_n)$$

6.3 Back Propogation

Now comes the final piece of the puzzle The question is, how do we calculate the partial derivatives for the weights? Back propogation is able to give us $\frac{d}{dW_{ij}^l} J(\theta, x, y)$ for every possible weight by re-using the activations calculated in the forward pass.

First we recall the gradient descent algorithm:

1. Initialize random weights $\boldsymbol{w}$, set regularization parameter λ, stopping condition ϵ, and training rate η
2. While the difference between weights from the previous step is less than ϵ:
    3. Randomly shuffle the training data rows
    4. For each training example input and output pair $(\boldsymbol{x}_n, y_n)$:
        5. Calculate the gradient vector $\frac{d}{d\boldsymbol{w}} J(\theta)$ for each weight
        6. Perform weight update: $\boldsymbol{w} = \boldsymbol{w} - \eta \frac{d\boldsymbol{e}_{L2}}{d\boldsymbol{w}}$

Now comes the calculation of those above gradients with back-propogation:

1. Perform a feedforward pass, computing actuvations for layers $L2, ..., L_n l$
2. For each output unit i in layer nl, set $\delta^{nl} = -(y_i - a_i^{nl}) f'(z_i^{nl})$
3. for all the subsequent layers, and for each node i in layer l: set $\delta_i^l = (\sum_{j=1}^{s_l + 1} W_{ji}^l \delta_j^{l+1}) f'(z_i^l)$
4. $\frac{d}{dW_{ij}^l} J(\theta, x, y) = a_j^l \delta_i^{l+1}$

6.4 Regularisation

There are a few different techniques for avoiding over fitting. Here they are:
Weight Decay: adding a component to the error term to account for the magnitude of weights.

Early Stopping: Stop with respect the smallest error of the validation set.

Both of these techniques (in the case of a quadratic regularisation term) have a similar effect in limiting the effective complexity of the model.

6.5 Autoencoders

Mapping inputs onto themselves. Autoencoders are effectively performing principal component analysis. In the case of linear activation functions, the error function has a unique global minimum.

Having a non linear activation function in the single middle hidden layer will not cause the autoencoder to perform non linear principal component analysis - instead, you need extra hidden layers surrounding the middle layer with non linear activations.

7 Big Data

7.1 Overview

Big data is useful for training models which can grow in complexity as the data size increases - for example, deep neural networks. Not all models will grow with the training size though - naive bayes is good on small training sets, but quickly reaches it's upper bound of performance as data is added.

The challenge though is training these models. Parallelisation can occur when we have multiple CPUs or computers (network latency being an issue in the later case).

Where algorithms contain sums over the data set, we can split up the computation as portions of the sum and compute each portion on a different machine.

7.2 Batch Gradient Descent

In English: in batch gradient descent, the gradient update is the sum of all individual weight updates (generally the partial derivatives of the zeroed error function). We can calculate that by giving the current model to each computer, but only subsets of the data to each computer. Each computer calculates its own error, then the master node sums them up to get the total weight update. All the models can then we updated.

Question: how to do this with stochastic gradient descent? Apparently it's some asynchronous update.

Algorithm 4 Map Reduce for Gradient Descent

procedure DISTRIBUTED GRADIENT DESCENT($\mathcal{D} := d_1, ..., d_m$) ▷ $\mathcal{D}$ is divided into M subsection
  $\mathbf{w} \leftarrow init()$ ▷ parameter vector
  while stopping condition not met **do**
    doInParallel($temp_m \leftarrow \delta\mathcal{E}_m(\mathbf{w})$)
    $\mathbf{w} \leftarrow \mathbf{w} + \sum_{m=1}^{M} temp_m$

7.3 Distributed K-Means

In English: here we calculate clusters for each subset of the data (as the average of the previously assigned points to that cluster) as well as the number of points assigned for that subset, then using all those averages (k * m, given k clusters and m machines) we can take a weighted average to get what would have been the overall average. The key here is that any correctly weighted average of a bunch of averages is the same as the average of all points :)

7.4 Distributed Expectation Maximisation for Gaussian Mixture Models

Similar to the above examples, but a little more complicated.

Algorithm 5 Map Reduce for Gradient Descent

procedure DISTRIBUTED GRADIENT DESCENT($\mathcal{D} := d_1, ..., d_m$) ▷ $\mathcal{D}$ is divided into M subsection

$\mu_{\mathbf{k}} \leftarrow initRandom(), \forall k$ ▷ init mu

while stopping condition not met **do**

$\mu'_{\mathbf{k}} \leftarrow 0, \forall k$ ▷ init mu

$\mathbf{n_k} \leftarrow 0, \forall k$ ▷ init n

▷ Map 5

$doInParallel_k(\mu_{\mathbf{km}} \leftarrow calcCluster(pts = D_m, cluster = k))$

$doInParallel_k(\mathbf{n_{km}} \leftarrow pointsInCluster(pts = D_m, cluster = k))$

▷ Reduce

$$\mu_k \leftarrow \frac{\sum_{m=1}^{M} \mu_{km}}{\sum_{m=1}^{M} n_{km}}$$

8 Misc

Example of partial derivative in linear regression with a non linear basis function vector.

Let's do an example where $\hat{\phi}$ is a vector of basis functions (same as ϕ bold), where $\phi_1 : f(x) = x^2$, $\phi_2 : f(x) = x^3$

In this case, ϕ acts on two inputs x_1 and x_2, along with weights w_1 and w_2 such that the linear regression model looks like:

$$y = w_1\phi_1(x_1) + w_2\phi_2(x_2)$$

If we wanted the derivative with respect of w_1 (for a single training example; we can easily expand it to be a sum), we treat all other components as constants and perform the chain rule like so:

$$\frac{d}{dw_1}\frac{1}{2}[t - \hat{w} \cdot \hat{\phi}(\hat{x})]^2$$

$$= \frac{d}{dw_1}\frac{1}{2}[t - w_1\phi_1 x_1 - w_2\phi_2 x_2]^2$$

$$= -(t - w_1\phi_1 x_1 - w_2\phi_2 x_2)\phi_1 x_1$$

9 Subject Summary

9.1 List of Algorithms

- Gradient Descent: batch, stochastic - proof from maximum likelyhood for regression, neural network version
- K-means
- EM: Soft, Hard, (and for GMM and Doc Clustering)

- Back-propogation
- Perceptron update
- Boot-strapping
- K-fold cross validation
- SGD

9.2 Exam Style Questions

Regression

- What are the advantages of sparse models in machine learning?
- Regularisation: what form of regularisation functions similarly to early stopping?
- Can L1 Regularisation be used in stochastic gradient desccent? If not, why not? Propose another regularisation technique and how it could be incorporated into SGD.
- Which of the following would constitute a linear regression model:
  1. $y = w_1 x_1 + w_0$
  2. $y = w_1 x_1^+ w_0$
  3. $y = w_1^2 x_1 + w_0$
  4. $y = w_1^2 x_1^2 + w_0^2$
  5. $y = sin(w_1 x_1) + w_0$
  6. $y = w_1 sin(x_1) + w_0$

Classification

- Write the formula which models the posterior in logistic regression.
- Show how the log likelyhood for logistic regression and how it derives to form the gradient update rule for SGD.
- describe the benefits of using a perceptron over mutliple binary classifiers for multi class classification.
- Logistic Regression is which type of probabilistic model? (Discriminative or Generative?)
- What are the detriments of generative models over discriminative models?

Unsupervised Learning

- Give formulas for the following in latent variable document clustering: word proportion parameter, document probabilitiy / mixing coefficients (ϕ)
- What is the mathematical/ algorithmic difference between hard and soft expectation maximisation?
- What types of data is K-means suitable for?

Neural Networks

- Autoencoders: under what circumstances does an autoencoder perform effective linear PCA vs non-linear PCA?
- Provide the derivative of the sigmoid activation function.
- For each of the following neural network applications, list the probability distribution which models the underlying problem, and the loss function you would use to train a neural network: regression, classification, multi-class classification.

Big Data

- Describe in general terms how any machine learning algorithm can be deployed in parallel.
- What over-head costs are associated with distributed computing versus single machine computing?
- Describe the map and reduce steps in the map reduce algorithm.

References

[1] B. K. Gholamreza Haffari, *Elements of Machine Learning*, 2018, vol. 19.