Machine Learning Notes

Chet Corcos

June 5, 2014

Contents

1 Conventions

All vectors are assumed to be vertical vectors and are denoted with bold-face, $\boldsymbol{v}$.
Matrices are capitalized and boldface, $\boldsymbol{M}$.

$p(\cdot)$ is the probability function.
$f(\cdot)$ is the prediction function, predicting some output, y given input parameters, $\boldsymbol{x}$. There are two types of prediction functions, continuous (regression) and classification.

$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$ is a dataset of N samples drawn from a joint distribution, $p(\boldsymbol{x}, y)$, with known inputs, $\boldsymbol{x}_i$ and corresponding correct output, y_i. $\boldsymbol{x} \sim \mathbb{R}^{D\times 1}$ corresponding to D dimensions or features of each sample. For predicting house prices, a feature may be size of the house, lot size, number of bathrooms, or number of bedrooms.

Often, data will be represented in matrix form where $\boldsymbol{X} \sim \mathbb{R}^{N\times D}$ and $\boldsymbol{y} \sim \mathbb{R}^{N\times 1}$. Sometimes this data may be preprocessed (zero mean, unit variance), and often, one of the dimensions will be a column of 1's representing a bias or offset term.

2 Probabilistic Machine Learning Terminology

2.1 Loss Function

$L(f(\boldsymbol{x}), y)$, is some measure of prediction error. There are many types of loss functions.

- L1-norm:	$L(f(\boldsymbol{x}), y) = \|f(\boldsymbol{x}) - y\|_1$	– city block distance
- L2-norm:	$L(f(\boldsymbol{x}), y) = \|f(\boldsymbol{x}) - y\|_2$	– just the euclidian
- p-norm:	$L(f(\boldsymbol{x}), y) = \|f(\boldsymbol{x}) - y\|_p$	– more generally speaking
- square-error:	$L(f(\boldsymbol{x}), y) = \|f(\boldsymbol{x}) - y\|_2^2$	– used in linear regression and logistic regression
- hinge-loss:	$L(f(\boldsymbol{x}), y) = \max(0, 1 - yf(\boldsymbol{x}))$	– used in SVMs for classification.
- exponential-loss:	$L(f(\boldsymbol{x}), y) = e^{-yf(\boldsymbol{x})}$	– used in adaboost for classification

The loss function is equal to the negated conditional log-likelihood

$$\begin{aligned} L(f(\boldsymbol{x}), y) &= -\log p(\mathcal{D}|\boldsymbol{w}) \\ &= -\sum_n \log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) \end{aligned}$$

2.2 Expected Conditional Risk

$$\begin{aligned} R(f, \boldsymbol{x}) &= \mathbb{E}_{y\sim p(y|\boldsymbol{x})}\, L(f(\boldsymbol{x}), y) \\ &= \int L(f(\boldsymbol{x}), y)\; p(y|\boldsymbol{x})\; dy \end{aligned}$$

says, given y follows some distribution conditioned on $\boldsymbol{x}$, what can we expect the loss to be if we marginalize over y. This means, what sort of loss (risk) can we expect from this predictor, $f(\cdot)$, given (conditioned) on the data we are given ($\boldsymbol{x}$).

2.3 Expected Risk

$$\begin{aligned} R(f) &= \mathbb{E}_{\boldsymbol{x}}\, R(f, \boldsymbol{x}) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\mathbb{E}_{y\sim p(y|\boldsymbol{x})}\, L(f(\boldsymbol{x}), y) \\ &= \int\int L(f(\boldsymbol{x}), y)\; p(y|\boldsymbol{x})p(\boldsymbol{x})\; dy\; d\boldsymbol{x} \\ &= \int\int L(f(\boldsymbol{x}), y)\; p(\boldsymbol{x}, y)\; dy\; d\boldsymbol{x} \end{aligned}$$

marginalizes the expected conditional risk over the input data, $\boldsymbol{x}$, leaving us with the overall expected risk of some prediction function.

2.4 Empirical Risk

Given some data, we can approximate the expected risk with the empirical risk given by

$$R_{\mathcal{D}}(f) = \frac{1}{N}\sum_{n} L(f(\boldsymbol{x}_n), y_n)$$

Also, with infinite data, empirical risk is the expected risk

$$\lim_{N\to\infty} R_{\mathcal{D}}(f) = R(f)$$

Emperical risk is by definition the average of the loss function over all data. This is also known as cross-entropy $\mathcal{E}(\cdot)$.

2.5 Bayes' Optimal Binary Classifier

This is a theoretical probabilistically optimal classifier. Assume some $\eta(\boldsymbol{x}) = p(y = 1|\boldsymbol{x})$. Then the Bayes' Optimal Binary Classifier is

$$f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \eta(\boldsymbol{x}) \geq 1/2 \\ 0 & \text{if } \eta(\boldsymbol{x}) < 1/2 \end{cases}$$

This is very useful in proving the performance of your classifier.

2.6 Bayes' Theorem

You are probably familiar with Bayes' Theorem but I want to go over it just to clear up some terms.

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\boldsymbol{w}) \times p(\mathcal{D}|\boldsymbol{w})}{p(\mathcal{D})}$$
$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

These terms will come up. Also pertinent to this topic is the concept of conjugate priors which defines the type of posterior distribution given the type of likelihood and prior distributions.

In plain english, this theorem describes that the probability of some parametric weights fitting some data is equal to any prior knowledge of what the weights should be times the likelihood that data is described by those weights divided by the probability of seeing that data.

When doing maximum likelihood estimation, we just maximize (typically the log) conditional likelihood. When we want to find the maximum a posterior solution, we typically leave out the evidence term because it is a constant that is nearly impossible to know and does not effect the optimization.

3 Optimization Background

3.1 Constrained Optimization – Lagrange Multipliers

Suppose we want to minimize some function subject to some constrains.

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t.}\ \ g(x) = 0 \end{aligned}$$

To solve this, we construct the *Lagrangian*.

$$L(x, \lambda) = f(x) + \lambda g(x)$$

Here, λ is called a *lagrange multiplier*. To solve our constrained minimization, take the derivative with respect to each of the *primal* variables, x and λ, and solve equal to zero. $\frac{d\, L(x,\lambda)}{dx} = 0$ and $\frac{d\, L(x,\lambda)}{d\lambda} = 0$. Substitute and solve.

If there is more than one constraint, just create another lagrange multiplier and add it to the lagrangian.

3.2 Lagrange Duality

Lagrange duality refers to the difference between the dual and primal formulations of an optimization problem.

Given the constrained optimization problem, the *primal* formulation of the problem is given by

$$\begin{aligned} \min_x \ & f(x) \\ \text{s.t.} \ & g_i(x) \leq 0 \ \ \forall \, i \end{aligned}$$

To get the *dual* formulation, we derive the lagrangian as follows, where $\{\lambda_i\}$ are the set of lagrange multipliers

$$L(x, \{\lambda_i\}) = f(x) + \sum_i \lambda_i g_i(x)$$

Note that

$$\begin{aligned} L(x, \{\lambda_i\}) &\leq f(x) \\ \min_x \max_{\{\lambda_i\}} L(x, \{\lambda_i\}) &= \min_x \ f(x) \text{ s.t. } g_i(x) \leq 0 \ \ \forall \, i \end{aligned}$$

Performing this min-max is difficult. This gives rise to the *dual* formulation which is a lower bound on the optimal solution.

$$\begin{aligned} g(\{\lambda_i\}) &= \min_x \ L(x, \{\lambda_i\}) \\ \max_{\{\lambda_i\}} g(\{\lambda_i\}) &= \max_{\{\lambda_i\}} \min_x \ L(x, \{\lambda_i\}) \end{aligned}$$

From before, we can clearly see that

$$g(\{\lambda_i\}) \leq f(x)$$

Thus the solution to the dual

$$\max_{\{\lambda_i\}} \ g(\{\lambda_i\}) \leq \min_x \ f(x)$$

The difference between the primal and dual solutions is called the duality gap.

The dual formulation is properly given by

$$\begin{aligned} \max_{\{\lambda_i\}} \ & g(\{\lambda_i\}) \\ s.t. \ & \lambda_i \geq 0 \ \ \forall \, i \end{aligned}$$

4 Nearest Neighbor Classifier (NNC)

This is a very basic classifier, but it has a very strong theoretical proof. We define the *nearest neighbor* as the sample in the training set with the smallest distance to some sample in question.

$$nn(\boldsymbol{x}) = \arg\min_n \ \text{distance}(\boldsymbol{x}, \boldsymbol{x}_n)$$

This distance function is typically the squared euclidian distance

$$nn(\boldsymbol{x}) = \arg\min_n \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$$

We then classify based on the classification of the nearest neighbor

$$f(\boldsymbol{x}) = y_{nn(\boldsymbol{x})}$$

This is called a non-parametric model because it depends only on our training data. We will have to carry our training data around with us in order to classify new samples.

4.1 K-Nearest Neighbors Classifier (KNN)

This simply extends nearest neighbor to classify based on the classes of the k nearest neighbors

$$knn(\boldsymbol{x}) = \{nn_1(\boldsymbol{x}), nn_2(\boldsymbol{x}), \ldots, nn_k(\boldsymbol{x})\}$$

We then classify based on the majority class $c \in \{C\}$ (reads: some class c that is an element of the set of all classes, $\{C\}$).

$$f(\boldsymbol{x}) = \arg\max_c \sum_{n \in knn(\boldsymbol{x})} \mathbb{I}(y_n == c)$$

Here, we utilize the identity function, $\mathbb{I}(\cdot)$ which returns 1 if input is true and 0 if the input is false.

4.2 Theoretical Guarantees

We have a simple loss function

$$L(f(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } f(\boldsymbol{x}) = y \\ 1 & \text{if } f(\boldsymbol{x}) \neq y \end{cases}$$

We can compute the expected conditional risk as follows. There are two possible ways of making a mistake and we can compute their probabilities (where $\eta(\boldsymbol{x}) = p(y = 1|\boldsymbol{x})$)
1) $p(f(\boldsymbol{x}) = 1|y = 0) = \eta(nn(\boldsymbol{x}))(1 - \eta(\boldsymbol{x}))$
2) $p(f(\boldsymbol{x}) = 0|y = 1) = (1 - \eta(nn(\boldsymbol{x})))\eta(\boldsymbol{x})$

Writing out the expected conditional risk

$$R(f, \boldsymbol{x}) = \eta(nn(\boldsymbol{x}))(1 - \eta(\boldsymbol{x})) + (1 - \eta(nn(\boldsymbol{x})))\boldsymbol{x})$$

It can be shown that

$$\lim_{N \to \infty} \eta(nn(\boldsymbol{x})) = \eta(\boldsymbol{x})$$

Thus for $N \to \infty$

$$R(f, \boldsymbol{x}) = 2\eta(\boldsymbol{x})(1 - \eta(\boldsymbol{x}))$$

When compared to the Bayes' optimal binary classifier

$$R(f^*, \boldsymbol{x}) = \min\{\eta(\boldsymbol{x}), 1 - \eta(\boldsymbol{x})\}$$

We see that

$$R(f, \boldsymbol{x}) = 2R(f^*, \boldsymbol{x})(1 - R(f^*, \boldsymbol{x}))$$

We can then proceed to show an upper bound on the expected risk

$$\begin{aligned}
R(f) &= \mathbb{E}_x R(f, \boldsymbol{x}) \\
&= \mathbb{E}_x 2R(f^*, \boldsymbol{x})(1 - R(f^*, \boldsymbol{x})) \\
&= 2\mathbb{E}_x R(f^*, \boldsymbol{x}) - 2\mathbb{E}_x R(f^*, \boldsymbol{x})^2 \\
&= 2R(f^*) - [2R(f^*)^2 + \text{var}(R(f^*, x))] \\
&= 2R(f^*)(1 - R(f^*)) - \text{var}(R(f^*, x)) \\
&\leq 2R(f^*)(1 - R(f^*))
\end{aligned}$$

In the last step, we drop off the variance term and set the $=$ to $\leq$ because the variance must be positive.

We have thus shown a very strong theoretical guarantee for NNC and that is

$$R(f^*) \leq R(f^{NNC}) \leq 2R(f^*, \boldsymbol{x})(1 - R(f^*, \boldsymbol{x}))$$

5 Linear Regression

Linear regression is the problem of fitting a line to a set of points given a square-error loss function and a linear parametric model defined by

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$$

where $\boldsymbol{w}$ are a set of parametric weights for each dimension.

Note the difference between parametric models (linear regression) and non-parametric models (KNN or NNC) is that non-parametric models use only the data to make predictions while parametric models make predictions based on some parameters, in this case $\boldsymbol{w}$. These parameters are often referred to as weights. This is because the prediction is based on a weighted sum of sample features (dimensions).

This gives us an empirical risk (or cross-entropy) of

$$R_{\mathcal{D}}(f) = \frac{1}{N} \sum_n \|y_n - f(\boldsymbol{x})\|_2^2$$

In this case, empirical risk is often referred to as the residual-sum-of-squares (RSS) or mean-square-error (MSE).

The proper probabilistic way of deriving the solution to the weights is by maximizing the conditional likelihood or probability of seeing the data. By doing , we are creating a *discriminative* model because we optimizing the conditional probability. Later on with Bayesian linear regression, we will create a *generative* model by maximizing the complete likelihood (joint probability).

$$\arg\max_{\boldsymbol{w}} p(\mathcal{D}|\boldsymbol{w}) = \arg\max_{\boldsymbol{w}} \prod_n p(y_n|x_n, \boldsymbol{w})$$

To clean up the notion, we will leave out the weights in the probability and assume it is implied

$$\arg\max_{\boldsymbol{w}} p(\mathcal{D}) = \arg\max_{\boldsymbol{w}} \prod_n p(y_n|x_n)$$

We assume a noisy observation model such that

$$y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \eta$$

where the noise, $\eta \sim N(0, \sigma^2)$ is zero mean gaussian noise. Thus for one training sample

$$\begin{aligned} p(y_n|x_n) &= N(y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}, \sigma^2) \\ &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})^2}{2\sigma^2}} \end{aligned}$$

Now we can maximize the conditional likelihood. A common trick is to equivalently maximize log-likelihood. This turns products into sums which makes things easier.

$$\begin{aligned} \log p(\mathcal{D}) &= \log \prod_n p(y_n|x_n) \\ &= \sum_n \log p(y_n|x_n) \\ &= \sum_n -\frac{(y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})^2}{2\sigma^2} - \log \sigma\sqrt{2\pi} \end{aligned}$$

Now, to find the optimal weights, we take the derivative set to zero

$$\frac{\partial}{\partial \boldsymbol{w}} \log p(\mathcal{D}) \propto \frac{\partial}{\partial \boldsymbol{w}} \sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})^2 = 0$$

This means that $\arg\max_{\boldsymbol{w}} \log p(\mathcal{D}) = \arg\min_{\boldsymbol{w}} R_{\mathcal{D}}(f)$!

$$
\begin{aligned}
\boldsymbol{w} &= \arg\min_{\boldsymbol{w}} R_{\mathcal{D}}(f) \\
&= \arg\min_{\boldsymbol{w}} \frac{1}{N} \sum_n (y_n - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)^{\mathrm{T}} (y_n - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \\
&= \arg\min_{\boldsymbol{w}} \frac{1}{N} \sum_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n \boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{w} - 2 y_n \boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{w} + y_n^2 \\
&= \arg\min_{\boldsymbol{w}} \frac{1}{N} \left[\boldsymbol{w}^{\mathrm{T}} \left(\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^{\mathrm{T}} \right) \boldsymbol{w} - 2 \left(\sum_n y_n \boldsymbol{x}_n^{\mathrm{T}} \right) \boldsymbol{w} \right] + \text{constant} \\
&= \arg\min_{\boldsymbol{w}} \frac{1}{N} \left[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{w} - 2 \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} \right) \boldsymbol{w} \right] + \text{constant} \\
0 &= \frac{\partial}{\partial \boldsymbol{w}} \frac{1}{N} \left[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{w} - 2 \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} \right) \boldsymbol{w} \right] \\
\boldsymbol{w}^{MLE} &= \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}
\end{aligned}
$$

The solution derived above is also known as the maximum likelihood estimation (MLE) solution or the least-mean-squares (LMS) solution.

Note that all of the different dimensions of the weights are independent. Also, the solution is independent of the noise, η.

Also note that we could use numerical optimization tools. What if D is large? That matrix inversion may be intractable. Thankfully, the (empirical risk) objective function is convex

$$
\begin{aligned}
\frac{\partial R_{\mathcal{D}}(f)}{\partial \boldsymbol{w}} &\propto \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{w} - 2 \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} \\
\frac{\partial^2 R_{\mathcal{D}}(f)}{\partial \boldsymbol{w} \boldsymbol{w}^{\mathrm{T}}} &= \boldsymbol{H} \\
&\propto \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}
\end{aligned}
$$

This is positive semidefinite, thus proving convexity.

Linear regression can be extended to nonlinear regression using a basis function, $\boldsymbol{\phi}(\boldsymbol{x})$, which transforms the data into a nonlinear basis. For example, given a dataset where $D = 2$, we can use a nonlinear basis function to transform the data into a quadratic space

$$
\boldsymbol{\phi}(\boldsymbol{x}) \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_1 \cdot x_2 \\ x_2^2 \end{bmatrix}
$$

This allows us to fit a quadratic to our data using linear regression rather than simply a line. In this way, we are increasing the dimensionality of our data. However, this will be prone to overfitting. See section 14 about the trade-off between model complexity and overfitting.

5.1 Ridge Linear Regression

Also known as regularized linear regression.

This can be thought of from two difference perspectives.

1) Given the MLE solution, what if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is not invertible? This could happen if $N < D$. Intuitively, this would mean there is not enough data to estimate all the parameters. This could also happen is the columns of $\boldsymbol{X}$ are not linearly independent. This would happen if any features are perfectly correlated.

An easy solution to this is to add a diagonal matrix to the solution

$$\boldsymbol{w} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

This makes linear regression more numerically stable. The matrix is now guaranteed to be invertible.

$\lambda > 0$ is called the regularization term. It is considered a *hyperparameter* of the model because it will need to be optimized separately from the optimization of the parametric weights of the model. For more on tuning hyperparameters, see section 16.

2) Suppose our model is susceptible to overfitting. Thus we want to our model to be more simple. We can do this by introducing a *prior* belief

$$p(\boldsymbol{w}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}{2\sigma^2}}$$

namely, that $\boldsymbol{w}$ is around zero resulting in a simple model. This line of thinking is to regard $\boldsymbol{w}$ as a random variable and we will use the observed data to update our *a prior* belief on $\boldsymbol{w}$.

Now, rather than maximizing the conditional likelihood, we want to maximize the joint probability

$$\begin{aligned}
p(\mathcal{D}, \boldsymbol{w}) &= p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) \\
\log p(\mathcal{D}, \boldsymbol{w}) &= \sum_n \log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) + \log p(\boldsymbol{w}) \\
&= -\frac{1}{2\sigma_0^2}\sum_n (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - y_n)^2 - \frac{1}{2\sigma^2}\|\boldsymbol{w}\|_2^2 + \text{constant} \\
\mathcal{E}(\boldsymbol{w}) &= \frac{1}{2\sigma_0^2}\sum_n (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - y_n)^2 + \frac{1}{2\sigma^2}\|\boldsymbol{w}\|_2^2
\end{aligned}$$

Where σ_0 is the variance of the noise. Now, we seek to maximize the *a posterior* (MAP) solution to the weights

$$\boldsymbol{w}^{MAP} = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D})$$

and we know that

$$\begin{aligned}
p(\boldsymbol{w}, \mathcal{D}) &= p(\boldsymbol{w}|\mathcal{D})p(\mathcal{D}) \\
\log p(\boldsymbol{w}, \mathcal{D}) &= \log p(\boldsymbol{w}|\mathcal{D}) + \log p(\mathcal{D}) \\
\log p(\boldsymbol{w}, \mathcal{D}) &= \log p(\boldsymbol{w}|\mathcal{D}) + \text{constant}
\end{aligned}$$

so maximizing the a posterior is analogous to maximizing the the joint likelihood. Thus,

$$\begin{aligned}
\boldsymbol{w}^{MAP} &= \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D}) \\
&= \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}, \mathcal{D}) \\
&= \arg\min_{\boldsymbol{w}} \mathcal{E}(\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum_n (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - y_n)^2 + \frac{\sigma_0^2}{\sigma^2}\|\boldsymbol{w}\|_2^2 \\
&= \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \frac{\sigma_0^2}{\sigma^2}\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}
\end{aligned}$$

where the regularization term $\lambda = \frac{\sigma_0^2}{\sigma^2}$, the ratio of the model noise variance σ_0, to the variance of the weights σ. A smaller σ indications a stronger prior for a simpler model, thus more regularization.

Again, regularization must be treated as a hyperparameter of the model. For more on tuning hyperparameters, see section 16.

5.2 Bayesian Linear Regression

This is the full blown Bayesian treatment of linear regression. Bayesian methods can be applied all over the place. Conjugate priors make life a bit easier though its not necessary. However, for linear regression, all distributions are Gaussian which makes things nice and easy.

First we define the likelihood as a normal distribution with some precision/variance due to noise

$$p(y|\boldsymbol{x}) = N(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}, \beta^{-1})$$

β is known as precision and is the inverse of the variance of the noise (remember $\eta \sim N(0, \sigma^2)$). We also define a prior that tries urge a simpler model.

$$p(\boldsymbol{w}) = N(\boldsymbol{0}, \alpha^{-1}\boldsymbol{I})$$

Our prior suggests that $\boldsymbol{w}$ is around zero resulting in a simple model. α is the precision that tells us how confident we think we are about where $\boldsymbol{w}$ is centered. Given this prior (the weights centered at zero), α defines how simple out model ought to be.

Now to derive the maximum a posterior solution

$$\begin{aligned}
p(\boldsymbol{w}|\mathcal{D}) &\propto p(\boldsymbol{w})p(y|\boldsymbol{x}, \boldsymbol{w}) \\
&\propto N(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}) \times \prod_n N(y_n|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n, \beta^{-1}) \\
&\propto \exp -\frac{\alpha}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} - \frac{\beta}{2}\sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)^2 \\
&\propto \exp -\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{w} - \boldsymbol{\mu}) \\
&\propto N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})
\end{aligned}$$

The last step involves completing the square.

$$\begin{aligned}\boldsymbol{\Sigma} &= \left(\alpha \boldsymbol{I} + \beta \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right)^{-1} \\ \boldsymbol{\mu} &= \beta \boldsymbol{\Sigma} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}\end{aligned}$$

Note that we derived what the MAP solution is proportional to. That is because the evidence is just a constant. If we were to maximize the log posterior probability, we would be left with $\boldsymbol{w} = \boldsymbol{\mu}$. This is the same solution as we saw before for ridge regression only this time we also have our confidence in that solution.

Next, we derive the predictive distribution for a new data point

$$\begin{aligned}p(y|\boldsymbol{x}, \mathcal{D}) &= \int \text{likelihood} \times \text{posterior}\ d\boldsymbol{w} \\ &= \int p(y|\boldsymbol{x}, \boldsymbol{w}) p(\boldsymbol{w}|\mathcal{D}) d\boldsymbol{w} \\ &\propto \int \exp -\frac{\beta}{2}(y - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})^2 - \frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{w} - \boldsymbol{\mu}) d\boldsymbol{w} \\ &\propto N(y|\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x}, \beta^{-1} + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{x}) \\ &\propto N(y|\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x}, \sigma^2(\boldsymbol{x})\end{aligned}$$

What we are left with is same prediction as before, $\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x}$, but now we also have a confidence for each data point, $\sigma^2(\boldsymbol{x})$! This allows you to plot your regression curve along with error bounds indicating confidence in any region of the curve.

So how do we choose α and β? There is no analytical solution. There are some iterative procedures involving eigendecomposition, but in practice, they are tuned as hyperparameters of the model using cross-validation.

A huge benefit of Bayesian linear regression is that we can update the model with new data without recalculating computing all of the training data. When we first train the model on the training data and get a posterior, we can use that as the prior to compute a new posterior with new data!

Another benefit is that we treat every prediction as an independent gaussian process and thus we can compute the confidence in each prediction given by $\sigma^2(\boldsymbol{x})$.

6 Logistic Regression

This is very analogous to linear regression except it is used for classification. It is a non-linear model with no analytical solution, but it is often referred to as a linear model because the decision boundary must be a hyperplane. The model returns a probability of some sample belonging to a specific class or not, defined by the conditional likelihood

$$\begin{aligned}p(y_n = c|\boldsymbol{x}_n) &= \frac{1}{1 + e^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n}} \\ &= \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)\end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function. Note that this is binary classification. Either $y = c$ or $y \neq c$. We will alter the notation to be $y = 1$ and $y = 0$ respectively. This allows us to rewrite this nicely

$$p(y_n|\boldsymbol{x}_n) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)^{y_n}(1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n))^{1-y_n}$$

We then proceed to determine the optimal weights using maximum likelihood estimation. But first, we take the log to make things easier

$$\log p(\mathcal{D}) = \sum_n y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n)\log(1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n))$$

It is convenient to work with the negation of the log likelihood known as the cross-entropy error function

$$\mathcal{E}(\boldsymbol{w}) = \sum_n -y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - (1 - y_n)\log(1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n))$$

We find the minimum of the cross-entropy error using the stationary point condition

$$\begin{aligned}\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} &= \sum_n -y_n[1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\boldsymbol{x}_n - (1 - y_n)\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)\boldsymbol{x}_n \\ 0 &= \sum_n [\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n]\boldsymbol{x}_n\end{aligned}$$

Note that there is no closed-form analytical solution to identify $\boldsymbol{w}$ from the stationary point condition. Thus we use numerical optimization. Gradient decent works, but second-order Newton's method works swimmingly. However inverting the hessian makes Newton's method unscalable. We can show that the cross-entropy function is convex, thus a numerical optimization can guarantee a global optima. We start by computing the hessian matrix

$$\begin{aligned}\boldsymbol{H} &= \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} \\ &= \sum_n \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)[1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\boldsymbol{x}_n\boldsymbol{x}_n^{\mathrm{T}} \\ &= \sum_n (\alpha_n\boldsymbol{x}_n)(\alpha_n\boldsymbol{x}_n)^{\mathrm{T}}\end{aligned}$$

where $\alpha_n = \sqrt{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)[1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]}$.

For any vector, $\boldsymbol{v}$,

$$\begin{aligned}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{v} &= \sum_n \boldsymbol{v}^{\mathrm{T}}(\alpha_n\boldsymbol{x}_n)(\alpha_n\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{v} \\ &= \sum_n [\alpha_n\boldsymbol{v}^{\mathrm{T}}\boldsymbol{x}_n]^2 \geq 0\end{aligned}$$

and thus, this optimization is convex which guarantees our solution is a global optima.

6.1 Multi-class Classification with Binary Logistic Regression

You have two choices for using binary classifiers for multi-class classification
1) "one vs rest": Train K classifiers, one for each class to discriminate between that class and the rest of the classes. On a new sample, predict with all K classifiers and choose the one with the highest probability. This method is beneficial if you have many classes.

2) "one vs one": Train "K choose 2" classifiers, one for each pair of classes to discriminate between them. On a new sample, predict with all classifiers and choose the class that was predicted most often. This method is beneficial if you have lot of data, because you are training on a subset of the data – only the data for the two classes.

6.2 Multinomial Logistic Regression

Multinomial logistic regression is used for multi-class classification and is a simple extension of logistic regression. The conditional likelihood is given by

$$p(y_n = c_k|\boldsymbol{x}_n) = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}_n}}{\sum_{k'} e^{\boldsymbol{w}_{k'}^{\mathrm{T}}\boldsymbol{x}_n}}$$

This is called the *softmax* function.

Also, since we are no longer doing binary classification, we need to use one-hot encoding for the target vector, $\boldsymbol{y}_n \sim \mathbb{R}^{K\times 1}$ such that

$$y_{nk} = \begin{cases} 1 & \text{if } y_n = k \\ 0 & \text{otherwise} \end{cases}$$

And now the conditional log likelihood

$$\begin{aligned}
\log P(\mathcal{D}) &= \sum_n \log P(\boldsymbol{y}_n|\boldsymbol{x}_n) \\
&= \sum_n \log \prod_k P(y_{nk} = 1|\boldsymbol{x}_n)^{y_{nk}} \\
&= \sum_n \sum_k y_{nk} \log P(y_{nk} = 1|\boldsymbol{x}_n) \\
\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) &= -\sum_n \sum_k y_{nk} \log P(y_{nk} = 1|\boldsymbol{x}_n)
\end{aligned}$$

The cross-entropy function is convex and can therefore has a unique global optimum. Optimization requires numerical techniques analogous to those used for binary logistic regression, but large-scale implementation of multinomial logistic regression is non-trivial both for the number of classes and the number of training samples.

7 Gaussian Discriminant Analysis (GDA)

Gaussian discriminant analysis is a generative classification model (funny, its called "discriminant" analysis). The primary benefit of GDA is it is a parametric classification model that has a closed form analytical solution, unlike logistic regression.

7.1 Linear Discriminant Analysis (LDA)

Linear discriminant analysis is very similar to logistic regression. Given a dataset with two classes (a binary classification problem), we great a generative model by fitting gaussians to the data. However, for LDA we must assume the same variance. Thus, we must must maximize the log likelihood to find the parameters $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$ and σ.

$$\log P(\mathcal{D}|\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \sigma) = \sum_{n:y_n=1} \log \left(p_1 \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\boldsymbol{x}_n - \boldsymbol{\mu}_1)^2}{2\sigma^2}} \right) + \sum_{n:y_n=2} \log \left(p_1 \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\boldsymbol{x}_n - \boldsymbol{\mu}_2)^2}{2\sigma^2}} \right)$$

Where $p_i = 1/N \sum_n \mathbb{I}(y_n == c_i) = N_i/N$. LDA is analogous to logistic regression because we will have a linear decision boundary given by

$$\begin{aligned} -\frac{(\boldsymbol{x} - \boldsymbol{\mu}_1)^2}{2\sigma^2} + \log p_1 &= -\frac{(\boldsymbol{x} - \boldsymbol{\mu}_2)^2}{2\sigma^2} + \log p_2 \\ \frac{\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1}{\sigma^2}\boldsymbol{x} + \left(\frac{\boldsymbol{\mu}_1^2 - \boldsymbol{\mu}_2^2}{2\sigma^2} - \log p_1 + \log p_2 \right) &= 0 \\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b} &= 0 \end{aligned}$$

If we decouple the bias term, $\boldsymbol{b}$, from $\boldsymbol{w}$ in logistic regression (remove the leading 1 from $\boldsymbol{x}$), such that $p(y_n = c|\boldsymbol{x}_n) = \sigma(\boldsymbol{b} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$ for logistic regression, then we can compute the equivalent logistic regression parameters

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \\ \boldsymbol{b} &= \frac{1}{2}(\boldsymbol{\mu}_1^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2) - \log \frac{p_2}{p_1} \end{aligned}$$

The benefit of LDA over logistic regression is that the parameters can be estimated analytically in closed-form. However, the closed-form LDA is not the optimal solution. In fact, LDA is rarely ever used in practice because constraining the variances to be the same is in fact a harder problem than QDA. LDA is explored simply to contrast with logistic regression. In practice, GDA usually refers to QDA.

7.2 Quadratic Discriminant Analysis (QDA)

Quadratic discriminant analysis is a generative model in the same way as LDA except we do not assume the covariances to be the same. This allows us to directly compute the parameters

of the gaussians for each class. Different covariance matrices gives rise to a quadratic decision boundary. If we do not assume the same covariances, the log likelihood decomposes into independent optimizations for generating a gaussian for each class!

In practice, for both LDA and QDA, you will never care to compute the decision boundary. You will approximate the data with gaussians, then compute the probability that new sample belongs to each gaussian (class), and choose the the class with the highest probability.

7.3 GDA vs Logistic Regression

GDA makes strong modeling assumptions and is more efficient both in amount of training data necessary and computation. If the modeling assumptions are correct (the data is gaussian) GDA is most certainly better. However logistic regression makes weaker assumptions and significantly more robust to deviations from modeling assumptions – it finds the optimal linear boundary. When data is non-gaussian, logistic regression will almost always be better. If the data is poisson, logistic regression is very good.

8 Kernel Methods

Kernel methods involve using a nonlinear basis function in your model and leveraging the *kernel trick*.

A kernel function $k(\cdot, \cdot)$ is the inner product between two nonlinear basis functions.

$$\begin{aligned} k(\boldsymbol{x}_1, \boldsymbol{x}_2) &= \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_2) \\ &= \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_1) \end{aligned}$$

The kernel matrix is a matrix of kernel functions defined by

$$\begin{aligned} \boldsymbol{K} &= \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} \\ &= \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \dots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \dots & \dots & \dots & \dots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \end{aligned}$$

The kernel matrix $\boldsymbol{K}$ is positive semi-definite and symmetric and $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ and doesn't depend on the dimension of the basis function!

Some other notation

$$\boldsymbol{\Phi}^{\mathrm{T}} = (\boldsymbol{\phi}(\boldsymbol{x}_1), \boldsymbol{\phi}(\boldsymbol{x}_2), \dots, \boldsymbol{\phi}(\boldsymbol{x}_N))$$

$\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ where M is the dimensionality of the basis function, $\boldsymbol{\phi}(\boldsymbol{x})$.

$$\begin{aligned}\boldsymbol{\Phi}\boldsymbol{x} &= \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}) \\ k(\boldsymbol{x}_2, \boldsymbol{x}) \\ \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}) \end{bmatrix} \\ &= \boldsymbol{k}_x\end{aligned}$$

What most compelling about the kernel trick is that since we do not need to compute the basis function to compute the kernel, we our kernel function can have an infinite dimensional basis function! Here is a simple example illustrating the point. Suppose we have the following mapping

$$\boldsymbol{\psi}_\theta(\boldsymbol{x}) = \begin{bmatrix} \cos(\theta x_1) \\ \sin(\theta x_1) \\ \cos(\theta x_2) \\ \sin(\theta x_2) \end{bmatrix}$$

Now lets consider the basis function

$$\boldsymbol{\phi}_L(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\psi}_0(\boldsymbol{x}) \\ \boldsymbol{\psi}_{\frac{2\pi}{L}}(\boldsymbol{x}) \\ \boldsymbol{\psi}_{2\frac{2\pi}{L}}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{\psi}_{L\frac{2\pi}{L}}(\boldsymbol{x}) \end{bmatrix}$$

where $L \in [0, 2\pi]$. Can we compute the inner product as $L \to \infty$?

$$\begin{aligned}\boldsymbol{\phi}_\infty(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}_\infty(\boldsymbol{x}) &= \lim_{L\to\infty} \boldsymbol{\phi}_L(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}_L(\boldsymbol{x}) \\ &= \int_0^{2\pi} \cos(\theta(x_{m1} - x_{n1})) + \cos(\theta(x_{m2} - x_{n2}))\, d\theta \\ &= 1 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} + 1 - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}\end{aligned}$$

This inner product of an infinite-dimensional feature space is finite and thus computable!

In practice, it is very difficult, if not impossible, to compute a basis function from a kernel function. A very popular kernel function is known as the Gaussian kernel, Radial Basis Function (RBF) kernel, or Gaussian RBF kernel

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = e^{-\|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2/2\sigma^2}$$

Composing new kernels is somewhat of a "black art" since we cannot back out the basis function. What is so beneficial about the kernel trick is it allows our model infinite complexity / dimensionality. The regularization term therefore regulates overfitting. The kernel trick is not possible without regularization.

8.1 Kernelized Ridge Regression

The cross-entropy error function for ridge regression with a nonlinear basis function is given by

$$\mathcal{E}(\boldsymbol{w}) = \frac{1}{2}\sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

And the $\boldsymbol{w}^{MAP}$ solution is given by

$$\begin{aligned}
\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} &= -\sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))\boldsymbol{\phi}(\boldsymbol{x}_n) + \lambda \boldsymbol{w} \\
\boldsymbol{w}^{MAP} &= \sum_n \frac{1}{\lambda}(y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \sum_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}
\end{aligned}$$

$\alpha_n = \frac{1}{\lambda}(y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))$ but we do not know the vector of all α_n's, $\boldsymbol{\alpha}$.

Next, we substitute this solution $\boldsymbol{w}^{MAP} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ back into $\mathcal{E}(\boldsymbol{w})$.

$$\begin{aligned}
\mathcal{E}(\boldsymbol{w}) &= \frac{1}{2}\sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 \\
\mathcal{E}(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - (\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} \\
&= \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}^2\boldsymbol{\alpha} - (\boldsymbol{K}\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha}
\end{aligned}$$

Note that we drop the $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y}$ because it is a constant and will not effect the cross-entropy minimization. Now lets derive the optimal $\boldsymbol{\alpha}$ from the cross-entropy error function using the stationary point condition

$$\begin{aligned}
\frac{\partial \mathcal{E}(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} &= \boldsymbol{K}^2\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} + \lambda\boldsymbol{K}\boldsymbol{\alpha} = 0 \\
\boldsymbol{\alpha} &= (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}
\end{aligned}$$

Note that the solution to $\boldsymbol{\alpha}$ depends on $\boldsymbol{K}$ and not directly on $\boldsymbol{\phi}(\boldsymbol{x})$! So long as you know how to compute the kernel function, you don't even need to know the basis function. More to come on this, but for now, we want back out the $\boldsymbol{w}^{MAP}$ solution

$$\boldsymbol{w}^{MAP} = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$$

Then, for prediction

$$\begin{aligned}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) &= \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K}+\lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{x} \\ &= \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K}+\lambda\boldsymbol{I})^{-1}\boldsymbol{k}_x\end{aligned}$$

Note that to make a prediction, once again, we only need to know the kernel function!

To summarize, first we must come up with a kernel function that satisfies

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_2, \boldsymbol{x}_1)$$

Then we can calculate $\boldsymbol{\alpha}$

$$\boldsymbol{\alpha} = (\boldsymbol{K}+\lambda\boldsymbol{I})^{-1}\boldsymbol{y}$$

And then we can make predictions

$$f(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K}+\lambda\boldsymbol{I})^{-1}\boldsymbol{k}_x$$

8.2 Kernelized Nearest Neighbor Classifier

Is $d(\boldsymbol{x}_m, \boldsymbol{x}_n) = \|\boldsymbol{x}_n - \boldsymbol{x}_m\|_2^2$ a kernel function?

$$\begin{aligned}d(\boldsymbol{x}_m, \boldsymbol{x}_n) &= \|\boldsymbol{x}_n - \boldsymbol{x}_m\|_2^2 \\ &= \boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_m + \boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_m - 2\boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_n \\ &= k(\boldsymbol{x}_m, \boldsymbol{x}_m) + k(\boldsymbol{x}_n, \boldsymbol{x}_n) - 2k(\boldsymbol{x}_m, \boldsymbol{x}_n)\end{aligned}$$

The summation of kernel functions and the product of kernel functions are also kernel functions. Thus, this distance is a kernel function! We have thus derived the kerneled nearest neighbor classifier as

$$y_n = \arg\min_n k(\boldsymbol{x}, \boldsymbol{x}_n)$$

And now you can use different kernel functions as well to transform the data into a different basis, perhaps an infinite basis! This can be easily extended for kernelized K-nearest neighbor classifier.

8.3 Gaussian Process

Gaussian process is the term given to kernelized Bayesian linear regression. We start by assuming the same prior as before

$$p(\boldsymbol{w}) \sim N(\boldsymbol{0}, \alpha^{-1}\boldsymbol{I})$$

This is equivalent to putting a prior on an infinite set of functions, $f_{\boldsymbol{w}}(\boldsymbol{x})$.

We use this same idea to derive the gaussian process, $\mathcal{GP}(\cdot)$.

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\cdot)$$

Here, were are saying that this function is a gaussian random process. This implies a joint distribution for every sample in the dataset

$$p(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_N)) \sim N(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{C}(\boldsymbol{x}))$$

Same as with Bayesian linear regression.

For simplicity, we choose $\boldsymbol{\mu}(\boldsymbol{x}) = \mathbf{0}$. Intuitively, as in Bayesian linear regression, the expected value of the prior's prediction is zero. $\boldsymbol{C}(\boldsymbol{x})$ is a covariance matrix. Namely, a kernel matrix!

Lacking a good derivation, here is the predictive distribution

$$p(y|\boldsymbol{x}, \mathcal{D}) = N(\boldsymbol{k}_{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{K}^{-1} \boldsymbol{y}, k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}_{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{K}^{-1} \boldsymbol{k}_{\boldsymbol{x}})$$

9 Support Vector Machines

Support vector machines are a kernelized method used for classification. Before discussing support vector machines, it helps to understand the perceptron.

9.1 Perceptron

The perceptron algorithm does binary classification. Suppose the two classes are $y_n \in \{-1, +1\}$ and we have a linear discriminant predictive function

$$f(\boldsymbol{x}_n) = \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$$

Our goal is to reduce the cross-entropy error function

$$\mathcal{E}(\boldsymbol{w}) = \sum_n \mathbb{I}(y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n))$$

The solution is to iterate through the data and

$$\begin{aligned} &\text{if } y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \quad \text{do nothing} \\ &\text{if } y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \quad \boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \boldsymbol{x}_n \end{aligned}$$

If the training data is linearly separable, the algorithm stops in a finite amount of time. Also, the parameter vector is always a linear combination of the training samples.

The problem with the perceptron is that it is not a high margin classifier. Once the algorithm converges, the decision boundary may be able separate the data perfectly, but that boundary is arbitrary and may come very close to misclassifying some sample by finding a sub-optimal solution. The solution to this is to not use a 1-0 loss function, rather a hinge-loss function.

9.2 Hinge-Loss

The hinge loss function ensures a margin for the linear discriminant (decision boundary).

$$\begin{aligned} L(f(\boldsymbol{x}), y) &= \begin{cases} 0 & \text{if } y\ f(\boldsymbol{x}) \geq 1 \\ 1 - y\ f(\boldsymbol{x}) & \text{otherwise} \end{cases} \\ &= \max(0, 1 - y\ f(\boldsymbol{x})) \end{aligned}$$

This is known as a high margin loss function and makes for a high margin classifier. This is because the hinge-loss function still penalizes the weights for samples that classified correctly but are close to the decision boundary.

9.3 SVM Derivation

SVMs do classification based on a hinge loss of a nonlinear basis function discriminant. The cross-entropy error function is defined by

$$\begin{aligned} \mathcal{E}(\boldsymbol{w}) &= \sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\ &= C\sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \end{aligned}$$

It is common to use $C = 1/\lambda$. We also rewrite with slack variables

$$\xi_n = \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b])$$

Using slack variables allows us to formulate a constrained differentiable convex optimization problem.

Now we are left with the *primal formulation* (note the simple change in notation – $f(\cdot)$ is the primal objective function).

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} f(\{\boldsymbol{x}_n\}) = \min_{\boldsymbol{w},b,\{\xi_n\}} & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ s.t.\ & 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \leq \xi_n,\ \ \forall n \\ & \xi_n \geq 0 \end{aligned}$$

To solve this constrained optimization problem, we introduce the Lagrangian. $\{\alpha_n\}$ and $\{\lambda_n\}$ are Lagrange multipliers ensuring the constraints.

$$\begin{aligned} & L(\{\boldsymbol{x}_n\}, \boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\ &= C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\left(1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\right) \\ &= C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n - \sum_n \alpha_n y_n \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) - \sum_n \alpha_n y_n b - \sum_n \alpha_n\xi_n \\ &= \sum_n (C - \lambda_n - \alpha_n)\xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_n \alpha_n - \sum_n \alpha_n y_n \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) - b\sum_n \alpha_n y_n \end{aligned}$$

To solve the primal formulation we need to perform a *max*, then a *min* on the Lagrangian. This is difficult.

$$
\begin{aligned}
L(\{\boldsymbol{x}_n\}, \boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) &\leq f(\{\boldsymbol{x}_n\}) \\
\min_{\boldsymbol{w},b,\{\xi_n\}} \max_{\{\alpha_n\},\{\lambda_n\}} L(\{\boldsymbol{x}_n\}, \boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) &= \min_{\boldsymbol{w},b,\{\xi_n\}} f(\{\boldsymbol{x}_n\})
\end{aligned}
$$

Flipping *min* and *max* gives rise to the dual formulation which gives yields (in most cases) a different result which is a lower bound on the optimal primal result (for more on lagrange duality, see section 3.2).

$$
\begin{aligned}
g(\{\alpha_n\}, \{\lambda_n\}) &= \min_{\boldsymbol{w},b,\{\xi_n\}} L(\{\boldsymbol{x}_n\}, \boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\
\max_{\{\alpha_n\},\{\lambda_n\}} g(\{\alpha_n\}, \{\lambda_n\}) &= \max_{\{\alpha_n\},\{\lambda_n\}} \min_{\boldsymbol{w},b,\{\xi_n\}} L(\{\boldsymbol{x}_n\}, \boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\})
\end{aligned}
$$

Note that

$$
\max_{\{\alpha_n\},\{\lambda_n\}} g(\{\alpha_n\}, \{\lambda_n\}) \leq \min_{\boldsymbol{w},b,\{\xi_n\}} f(\{\boldsymbol{x}_n\})
$$

Where $g(\{\alpha_n\}, \{\lambda_n\})$ is the dual objective function and $f(\{\boldsymbol{x}_n\})$ is the primal objective function. This discrepancy is called the duality gap.

Continuing to derive the dual formulation, we minimize L using the stationary point condition with respect to the primal variables

$$
\begin{aligned}
\frac{dL}{d\boldsymbol{w}} &= \boldsymbol{w} - \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) = 0 \\
\frac{dL}{db} &= \sum_n \alpha_n y_n = 0 \\
\frac{dL}{d\xi_n} &= C - \alpha_n - \lambda_n = 0
\end{aligned}
$$

Note here that one of the constraints solves for the primal weight variable. Substitute the back to find the dual objective function:

$$
\begin{aligned}
g(\{\alpha_n\}, \{\lambda_n\}) &= \min_{\boldsymbol{w},b,\{\xi_n\}} L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\
&= \cancel{\sum_n (C - \lambda_n - \alpha_n)\xi_n} + \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_n \alpha_n - \sum_n \alpha_n y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) - b \cancel{\sum_n \alpha_n y_n} \\
&= \frac{1}{2}\| \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)\|_2^2 + \sum_n \alpha_n - \sum_n \alpha_n y_n \left(\sum_m y_m \alpha_m \boldsymbol{\phi}(\boldsymbol{x}_m) \right)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \sum_n \alpha_n - \frac{1}{2} \sum_{mn} \alpha_n \alpha_m y_n y_m \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)
\end{aligned}
$$

Now we have our dual formulation subject to the Lagrangian constraints we previously derived:

$$\begin{aligned}\max_{\{\alpha_n\}\in\mathbb{R}^+,\{\lambda_n\}\in\mathbb{R}^+} & g(\{\alpha_n\},\{\lambda_n\}) = \sum_n \alpha_n - \frac{1}{2}\sum_{mn}\alpha_n\alpha_m y_n y_m \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) \\ s.t. \quad & \alpha_n \geq 0 \ \ \forall n \\ & \lambda_n \geq 0 \ \ \forall n \\ & \sum_n \alpha_n y_n = 0 \\ & C - \alpha_n - \lambda_n = 0 \ \ \forall n\end{aligned}$$

We can clean this up a little bit

$$\begin{aligned} \lambda_n &\geq 0 \\ C - \alpha_n - \lambda_n &= 0 \\ \lambda_n &= C - \alpha_n \\ \implies \alpha_n &\leq C \end{aligned}$$

The final form of the the dual formulation is

$$\begin{aligned}\max_{\{\alpha_n\},\{\lambda_n\}} & \sum_n \alpha_n - \frac{1}{2}\sum_{mn}\alpha_n\alpha_m y_n y_m k(\boldsymbol{x}_m,\boldsymbol{x}_n) \\ s.t. \quad & 0 \leq \alpha_n \leq C \ \ \forall\, n \\ & \sum_n \alpha_n y_n = 0\end{aligned}$$

One obvious benefit of the dual formulation is the use of a kernel function! This is a quadratic programming problem and can be solved using MATLAB's *quadprog()* function.

9.4 Analysis

One of the primal variables has already been derived while solving the lagrangian, that is

$$\boldsymbol{w} = \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

To solve for b, we look at complementary slackness. Complementary slackness says the lagrange multipliers times the constrains in the lagrangian must equal zero at the optimal solution to both primal and dual

$$\begin{aligned} &\lambda_n \xi_n = 0 \\ &\alpha_n \left(1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\right) = 0 \end{aligned}$$

From the first condition, $\alpha_n < C$,

$$\begin{aligned} \lambda_n = C - \alpha_n > 0 \\ \implies \xi_n = 0 \end{aligned}$$

And if we assume that $\alpha_n \neq 0$, then

$$\begin{aligned} 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n = 0 \\ \implies b = y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) \end{aligned}$$

for $y \in \{-1, +1\}$.

From this analysis, we can deduce from $\boldsymbol{w} = \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)$ that the solution is dependent only those samples whose corresponding $\alpha_n > 0$. These samples are called *support vectors*. This allows us to throw out all of the training data for those $\alpha_n = 0$, shrinking the amount of data we need to carry around.

From complementary slackness we see that for those support vectors ($\alpha_n > 0$):
- if $\xi_n = 0$, then $y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$. These are points are correctly classified and are right on the hinge of the loss function, or $1/\|\boldsymbol{w}\|_2$ away from the decision boundary.
- if $\xi_n < 1$, then $y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] > 0$. These points are classified correctly, but are not outside the margin of the hinge-loss function.
- if $\xi_n > 1$, then $y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] < 0$. These points are misclassified.

As you can see, the support vectors are only those samples that are non-zero in the hinge-loss function. This means samples that are misclassified or within the margin of the decision boundary.

To predict the classification of a new sample,

$$\begin{aligned} y &= \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b) \\ &= \text{sign}\left(\left[\sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)\right]^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) + b\right) \\ &= \text{sign}\left(\sum_n y_n \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}) + b\right) \end{aligned}$$

10 Adaboost

Adaboost stands for adaptive boosting. Boosting combines a lot of classifiers in a greedy way to construct a more powerful classifier and more complex decision boundaries. Since boosting creates a cascade of classifiers, it is best to use weak classifiers that are quick and easy to solve.

Our boosting algorithm prediction function will be

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_t \beta_t h_t(\boldsymbol{x})\right)$$

where β_t a weight on each weak classifier $h_t(\boldsymbol{x})$.

Here's how it works. Every data sample has a weighted exponential-loss. Initially, all of their weights are the same, $w_1(n) = 1/N$. Train a weak classifier, $h_t(\boldsymbol{x})$ by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}(y_n \neq h_t(\boldsymbol{x}_n))$$

Calculate the weight of this classifier

$$\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$$

Update the weights on each data sample based on the weighted exponential-loss

$$w_{t+1}(n) \leftarrow w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

Then normalize the them

$$w_{t+1}(n) \leftarrow \frac{w_{t+1}(n)}{\sum_n w_{t+1}(n)}$$

Now, the sample weights are exponentially weighted so that misclassified samples are weighted more. This is called a greedy algorithm. Loop back to training a new weak classifier, $h_{t+1}(\boldsymbol{x})$. Continue until convergence.

11 K-Means Clustering

K-means is an unsupervised learning algorithm that attempts to cluster data into K clusters. The objective function we wish to minimize is

$$J = \sum_n \sum_k r_{nk}\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2^2$$

where $\boldsymbol{\mu}_k$ is the centroid of each cluster and $r_{nk} \in \{0, 1\}$ is an indicator variable where $r_{nk} = 1$ if and only if $y_n = k = \arg\min_k \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2$, sample n is closest to centroid k.

This algorithm works by initializing the $\boldsymbol{\mu}_k$'s randomly. Determine the classification of all samples, r_{nk}. Then update $\boldsymbol{\mu}_k$ to be the centroid of all the samples belonging to cluster k. Loop back to re-classifying all of the samples again. Iterate until convergence.

Note that K-means does not guarantee the procedure terminates at a global optimum. In practice, K-means is run multiple times and the best solution is used.

Also, determining K is non-trivial. It is a hyperparameter, but it does not suffice to do cross-validation because the optimal $K = N$.

12 Gaussian Mixture Model (GMM)

GMM is a generative unsupervised clustering algorithm. Suppose we have a dataset made of up of multiple gaussian distributions, but they overlap. K-means does not return a probability of belonging to each cluster, only a classification. GMMs try to solve this problem by estimating the gaussians and returning probabilities of belonging to each cluster.

A GMM has the following density function

$$p(\boldsymbol{x}) = \sum_k w_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Each gaussian has a mean, covariance and weight associated with that cluster, w_k. Because $p(\boldsymbol{x})$ is a probability, $w_k \geq 0 \; \forall \; k$ and $\sum_k w_k = 1$.

This optimization is going to be tricky because of the constraints on the parameters. Namely, that $\sum_k w_k = 1$ and $\boldsymbol{\Sigma}_k$ must be positive semi-definite.

So, lets suppose we know the classification of each. Let z_n denote the classification for each sample. Thus $w_k = p(z = k) = p(z_n = k) \; \forall \; n$. Now consider the joint distribution

$$p(\boldsymbol{x}, z) = p(z)p(\boldsymbol{x}|z)$$

Then the conditional distribution is

$$p(\boldsymbol{x}|z = k) = N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

and the marginal distribution is

$$p(\boldsymbol{x}) = \sum_k w_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

The *complete* likelihood is

$$\begin{aligned}\sum_n \log p(\boldsymbol{x}_n, z_n) &= \sum_n \log p(z_n)p(\boldsymbol{x}_n|z_n) \\ &= \sum_k \sum_{n:z_n=k} \log p(z_n)p(\boldsymbol{x}_n|z_n)\end{aligned}$$

Let is define $\gamma_{nk} \in \{0, 1\}$ to indicate whether $z_n = k$. Then we can write

$$\begin{aligned}\sum_n \log p(\boldsymbol{x}_n, z_n) &= \sum_k \sum_n \gamma_{nk}[\log w_k + \log N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)] \\ &= \sum_k \sum_n \gamma_{nk} \log w_k + \sum_k \left[\sum_n \gamma_{nk} \log N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right]\end{aligned}$$

After taking the derivative and solving the maximum likelihood estimation, we come to a rather intuitive solution

$$
\begin{aligned}
w_k &= \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}} \\
\boldsymbol{\mu}_k &= \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \boldsymbol{x}_n \\
\boldsymbol{\Sigma}_k &= \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}
\end{aligned}
$$

This is nice and all, but we aren't given z_n, so we can compute them given the posterior probability

$$
\begin{aligned}
p(z_n = k|\boldsymbol{x}_n) &= \frac{p(\boldsymbol{x}_n|z_n = k)p(z_n = k)}{p(\boldsymbol{x}_n)} \\
&= \frac{p(\boldsymbol{x}_n|z_n = k)p(z_n = k)}{\sum_{k'} p(\boldsymbol{x}_n|z_n = k')p(z_n = k')}
\end{aligned}
$$

Note that we need to know all the parameters to be able to calculate the posterior $p(z_n = k|\boldsymbol{x}_n)$.

To get around this, first we are going to assume a *soft* γ_{nk} meaning that rather than γ_{nk} being binary, it is the posterior probability

$$
\gamma_{nk} = p(z_n = k|\boldsymbol{x}_n)
$$

Now we can come up with a simple iterative algorithm to find a solution. First initialize random parameters $\theta = \{\{\boldsymbol{\mu}_k\}, \{\boldsymbol{\Sigma}_k\}, \{w_k\}\}$. Then we compute the γ_{nk}'s. Then we update θ based on the new γ_{nk}'s and loop back over. Note that this optimization is not convex. This solution can be derived from the expectation maximization algorithm.

12.1 Expectation Maximization (EM) Algorithm

In general, EM is used to estimate parameters for probabilistic models with hidden/latent variables

$$
p(x|\theta) = \sum_z p(x, z|\theta)
$$

where x is observed, θ are the model parameters, and z is hidden. To obtain the maximum likelihood estimate of θ

$$
\begin{aligned}
\theta &= \arg\max_\theta l(\theta) \\
&= \arg\max_\theta \sum_n \log p(x_n|\theta) \\
&= \arg\max_\theta \sum_n \log \sum_{z_n} p(x_n, z_n|\theta)
\end{aligned}
$$

$l(\theta)$ is called the *incomplete* log-likelihood. The difficulty with the incomplete log-likelihood is that it needs to sum over all possible hidden variables and then take the logarithm. This log-sum format

makes computation intractable. Thus the EM algorithm leverages a clever trick to change this into sum-log by changing this into the expected (*complete*) log-likelihood

$$\begin{aligned} Q_q(\theta) &= \sum_n \mathbb{E}_{z_n \sim q(z_n)} \log p(x_n, z_n|\theta) \\ &= \sum_n \sum_{z_n} q(z_n) \log p(x_n, z_n|\theta) \end{aligned}$$

Now if we choose the distribution of z to be the posterior distribution, $q(z) = p(z|x, \theta)$, then we define

$$\begin{aligned} Q(\theta) &= Q_{z \sim p(z|x,\theta)}(\theta) \\ &= \sum_n \sum_{z_n} p(z|x, \theta) \log p(x_n, z_n|\theta) \\ &= \sum_n \sum_{z_n} p(z|x, \theta)[\log p(x_n|\theta) + \log p(z_n|x_n, \theta)] \\ &= \sum_n \sum_{z_n} p(z|x, \theta) \log p(x_n|\theta) + \sum_n \sum_{z_n} p(z|x, \theta) \log p(z_n|x_n, \theta) \\ &= \sum_n \log p(x_n|\theta) \sum_{z_n} p(z|x, \theta) + \sum_n \mathbb{H}[p(z|x, \theta)] \\ &= \sum_n \log p(x_n|\theta) + \sum_n \mathbb{H}[p(z|x, \theta)] \\ &= l(\theta) + \sum_n \mathbb{H}[p(z|x, \theta)] \end{aligned}$$

Where $\mathbb{H}[p(x)] = -\int p(x) \log p(x) dx$ is known as the entropy of the probabilistic distribution, $p(x)$. As before, we need to know the parameters, θ, to compute the posterior probability $p(z|x, \theta)$. Thus we will use a known θ^{OLD} to compute the expected likelihood.

$$Q(\theta, \theta^{OLD}) = \sum_n \sum_{z_n} p(z|x, \theta^{OLD}) \log p(x_n, z_n|\theta)$$

It can be shown that

$$\begin{aligned} l(\theta) &\geq Q(\theta, \theta^{OLD}) + \sum_n \mathbb{H}[p(z|x, \theta^{OLD})] \\ &\geq A(\theta, \theta^{OLD}) \end{aligned}$$

and thus we have a lower lower bound on the log-likelihood defined by the *auxiliary function*, $A(\theta, \theta^{OLD})$. An important property of the auxiliary function is that $A(\theta, \theta) = l(\theta)$.

Thus we want to maximize the auxiliary function

$$\theta^{NEW} = \arg\max_\theta A(\theta, \theta^{OLD})$$

and repeat this process such that

$$\theta^{(t+1)} \leftarrow \arg\max_\theta A(\theta, \theta^{(t)})$$

However, this maximization step does not depend on the entropy term, so

$$\theta^{(t+1)} \leftarrow \arg\max_\theta Q(\theta, \theta^{(t)})$$

Thus, the EM algorithm procedure iteratively predicts the posterior probability (the E-step),

$$p(z|x, \theta^{OLD})$$

and then updated the parameters in by the maximization (the M-step)

$$\theta^{(t+1)} \leftarrow \arg\max_\theta Q(\theta, \theta^{(t)})$$

The EM algorithm converges, but only to a local optima – a global optima is not guaranteed to be found.

12.2 EM for GMM

For the E-step, we simply compute

$$\begin{aligned}
\gamma_{nk} &= p(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}) \\
&= \frac{p(\boldsymbol{x}_n|z_n = k, \boldsymbol{\theta})p(z_n = k)}{\sum_{k'} p(\boldsymbol{x}_n|z_n = k', \boldsymbol{\theta})p(z_n = k')} \\
&= \frac{w_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'} w_{k'} N(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})}
\end{aligned}$$

Then for the M-step

$$\begin{aligned}
Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) &= \sum_n \sum_k p(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^{OLD}) \log p(\boldsymbol{x}_n, z_n = k|\boldsymbol{\theta}) \\
&= \sum_n \sum_k \gamma_{nk} \log p(\boldsymbol{x}_n, z_n = k|\boldsymbol{\theta}) \\
&= \sum_n \sum_k \gamma_{nk} \log p(\boldsymbol{x}_n|z_n = k, \boldsymbol{\theta})\ p(z_n = k|\boldsymbol{\theta}) \\
&= \sum_n \sum_k \gamma_{nk} \log N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\ \omega_k \\
&= \sum_n \sum_k \gamma_{nk} \log N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \gamma_{nk} log\ \omega_k
\end{aligned}$$

To find the optimal $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, we use the stationary point condition

$$
\begin{aligned}
\frac{d}{d\boldsymbol{\mu}_k} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) &= \frac{d}{d\boldsymbol{\mu}_k} \sum_n \sum_k \gamma_{nk} log\ N(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \gamma_{nk} log\ \omega_k \\
0 &= \frac{d}{d\boldsymbol{\mu}_k} \sum_n \sum_k \gamma_{nk} log\ \left((2\pi)^{-d/2} |\boldsymbol{\Sigma}_k|^{-1/2} \exp\{ -\frac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \} \right) \\
0 &= \frac{d}{d\boldsymbol{\mu}_k} \sum_n \sum_k -\gamma_{nk} \frac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \\
0 &= \sum_n \gamma_{nk} \frac{1}{2} \boldsymbol{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \\
\sum_n \gamma_{nk} \boldsymbol{\mu}_k &= \sum_n \gamma_{nk} \boldsymbol{x}_n \\
\boldsymbol{\mu}_k &= \frac{\sum_n \gamma_{nk} \boldsymbol{x}_n}{\sum_n \gamma_{nk}} \\
\boldsymbol{\mu}_k &= \frac{1}{N_k} \sum_n \gamma_{nk} \boldsymbol{x}_n
\end{aligned}
$$

$$
\begin{aligned}
\frac{d}{d\boldsymbol{\Sigma}_k} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) &= \frac{d}{d\boldsymbol{\Sigma}_k} \sum_n \sum_k \gamma_{nk} log\ N(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \gamma_{nk} log\ \omega_k \\
0 &= \frac{d}{d\boldsymbol{\Sigma}_k} \sum_n \sum_k \gamma_{nk} log\ \left((2\pi)^{-d/2} |\boldsymbol{\Sigma}_k|^{-1/2} \exp\{ -\frac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \} \right) \\
0 &= \frac{d}{d\boldsymbol{\Sigma}_k} \sum_n \sum_k \gamma_{nk} log\ |\boldsymbol{\Sigma}_k|^{-1/2} - \frac{d}{d\boldsymbol{\Sigma}_k} \sum_n \sum_k \gamma_{nk} \frac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \\
0 &= \frac{d}{d\boldsymbol{\Sigma}_k} \sum_n \sum_k \gamma_{nk} log\ |\boldsymbol{\Sigma}_k| + \frac{d}{d\boldsymbol{\Sigma}_k} \sum_n \sum_k \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \boldsymbol{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \\
0 &= \left(\boldsymbol{\Sigma}_k^{-1} \right)^{\mathrm{T}} \sum_n \gamma_{nk} + \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \\
0 &= \boldsymbol{\Sigma}_k N_k + \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \\
\boldsymbol{\Sigma}_k &= \frac{1}{N_k} \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}
\end{aligned}
$$

To find the optimal ω_k we must use Lagrange to constrain $\sum_k \omega_k = 1$.

$$
\mathcal{L}(\lambda, \boldsymbol{\theta}) = Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) - \lambda \left(\sum_k \omega_k - 1 \right)
$$

And now we solve for the optimal parameters

$$\begin{aligned}
\frac{d\mathcal{L}(\lambda, \boldsymbol{\theta})}{d\omega_k} &= \frac{d}{d\omega_k}\left[\sum_n \sum_k \gamma_{nk} \left(log\ N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + log\ \omega_k\right) - \lambda\left(\sum_k \omega_k - 1\right)\right] \\
0 &= \frac{\sum_n \gamma_{nk}}{\omega_k} - \lambda \\
\omega_k &= \frac{\sum_n \gamma_{nk}}{\lambda} \\
\sum_k \omega_k &= 1 \\
1 &= \sum_k \frac{\sum_n \gamma_{nk}}{\lambda} \\
\lambda &= \sum_k \sum_n \gamma_{nk} \\
\omega_k &= \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}} \\
\omega_k &= \frac{N_k}{\sum_k N_k}
\end{aligned}$$

13 Dimensionality Reduction

Very high dimensional data will make algorithms slower and even intractable.

13.1 Principle Component Analysis (PCA)

Often, data may be highly dimensional but highly correlated. If there are correlated features, it is beneficial to reduce the dimensionality of the data to have only uncorrelated features.

One way of deriving PCA is to maximize the projected variance in the data. This derivation assumes the data has zero-mean and the projection vector is given by $\boldsymbol{u}$.

Variance is given by (given the centered data assumption)

$$\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$$

Thus the formulation is

$$\max_{\boldsymbol{u}} \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{u} = \max_{\boldsymbol{u}} \frac{1}{N}\boldsymbol{u}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u}$$

but we must constrain $\boldsymbol{u}$ so that it doesn't become arbitrarily large. Thus

$$\begin{aligned}
\max_{\boldsymbol{u}}\ & \frac{1}{N}\boldsymbol{u}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u} \\
\text{s.t.}\ & \|\boldsymbol{u}\|_2^2 = 1
\end{aligned}$$

Solve using the lagrangian

$$\mathcal{L}(\lambda, \boldsymbol{u}) = \frac{1}{N}\boldsymbol{u}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u} + \lambda(1 - \boldsymbol{u}^{\mathrm{T}}\boldsymbol{u})$$

when solving for the optima, we see that $(\lambda, \boldsymbol{u})$ is an eigenvalue-eigenvector pair!

$$\begin{aligned}\frac{\partial \mathcal{L}(\lambda, \boldsymbol{u})}{\partial \boldsymbol{u}} &= 2\boldsymbol{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} = 0 \\ \boldsymbol{\Sigma}\boldsymbol{u} &= \lambda\boldsymbol{u}\end{aligned}$$

plugging back to the original formulation

$$\max_{\boldsymbol{u}} \boldsymbol{u}^{\mathrm{T}}\lambda\boldsymbol{u}$$

We see that $\boldsymbol{u}$ must be the eigenvector with the largest eigenvalue.

Its not a stretch to see that to project into multiple dimensions, D, with maximal projected variance we will project using the eigenvectors associated with the D largest eigenvalues.

Note that if you do not center the data, you will not have maximized the variance. However, funny enough, it still sort of works though you will not get the same result as PCA. In fact, your first axis will likely be very close to the mean vector.

13.2 Kernelized Principle Component Analysis (kPCA)

Starting from where we left off with standard PCA

$$\begin{aligned}\boldsymbol{\Sigma}\boldsymbol{u} &= \lambda\boldsymbol{u} \\ \frac{1}{N}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u} &= \lambda\boldsymbol{u} \\ \boldsymbol{u} &= \frac{1}{\lambda N}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u} \\ &= \boldsymbol{X}^{\mathrm{T}}\left(\frac{1}{\lambda N}\boldsymbol{X}\boldsymbol{u}\right) \\ &= \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha}\end{aligned}$$

What is $\boldsymbol{\alpha}$? Plugging into for $\boldsymbol{u} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha}$

$$\begin{aligned}\frac{1}{N}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha} &= \lambda\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha} \\ \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha} &= \lambda\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\alpha}\end{aligned}$$

Now replace $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ with a kernel matrix, $\boldsymbol{K}$ and you have

$$\begin{aligned}\frac{1}{N}\boldsymbol{K}\boldsymbol{K}\boldsymbol{\alpha} &= \lambda\boldsymbol{K}\boldsymbol{\alpha} \\ \frac{1}{N}\boldsymbol{K}\boldsymbol{\alpha} &= \lambda\boldsymbol{\alpha}\end{aligned}$$

$\boldsymbol{\alpha}$ is the eigenvector with the largest associated eigenvalue of the kernel matrix!

Subtle Issue 1: the kernel matrix must be centralized. This can be done like so

$$\boldsymbol{K}_{centered} = (\boldsymbol{I} - \boldsymbol{1})\boldsymbol{K}(\boldsymbol{I} - \boldsymbol{1})$$

Subtle Issue 2: to ensure $\|\boldsymbol{u}\|_2^2 = 1$, we must rescale $\boldsymbol{\alpha}$ by $\frac{1}{\sqrt{\lambda N}}$.

14 Bias vs Variance Trade-off

Bias vs Variance is the trade off between model complexity, over-fitting and under-fitting. There are some strong theoretical proofs as well.

Lets assume a square-error loss function, and lets assume our prediction function, $f(\cdot)$, is trained on the data, $\mathcal{D}$, and thus is denoted $f_{\mathcal{D}}(\cdot)$. So, the expected risk is defined by

$$\begin{aligned} R(f_{\mathcal{D}}) &= \int\int L(f_{\mathcal{D}}(\boldsymbol{x}), y)\; p(\boldsymbol{x}, y)\; dy\; d\boldsymbol{x} \\ &= \int\int (f_{\mathcal{D}}(\boldsymbol{x}) - y)^2\; p(\boldsymbol{x}, y)\; dy\; d\boldsymbol{x} \end{aligned}$$

Lets define the averaged risk

$$\mathbb{E}_{\mathcal{D}} R(f_{\mathcal{D}}) = \int\int\int (f_{\mathcal{D}}(\boldsymbol{x}) - y)^2\; p(\boldsymbol{x}, y)\; dy\; d\boldsymbol{x}\; P(\mathcal{D})\; d\mathcal{D}$$

This marginalized out the randomness with respect to the data, $\mathcal{D}$ and the risk. Lets also define the averaged prediction

$$\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) = \int f_{\mathcal{D}}(\boldsymbol{x}) P(\mathcal{D})\; d\mathcal{D}$$

Again, to marginalize out the randomness of the data on training the model.

Now, lets add and subtract the averaged prediction

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}} R(f_{\mathcal{D}}) &= \int\int\int (f_{\mathcal{D}}(\boldsymbol{x}) - y)^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&= \int\int\int (f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) + \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y)^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&= \int\int\int (f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}))^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&\quad + \int\int\int (\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y)^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&\quad + \int\int\int (f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}))(\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y) \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D}
\end{aligned}$$

The third term equals zero

$$\begin{aligned}
&\int\int\int (f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}))(\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y) \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&\int\int \cancel{\left[\int (f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x})) \; P(\mathcal{D}) \; d\mathcal{D}\right]} (\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y) \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x}
\end{aligned}$$

So now we are left with the two terms

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}} R(f_{\mathcal{D}}) &= \int\int\int [f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x})]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&\quad + \int\int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D}
\end{aligned}$$

Lets explore the second term further. It is no longer dependent on $\mathcal{D}$ so lets get rid of that.

$$\begin{aligned}
&\int\int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&= \int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x}
\end{aligned}$$

To simplify further, we will use a similar trick by defining the the averaged target

$$\mathbb{E}_y y = \int y \; p(y|\boldsymbol{x}) \; dy$$

Plug it in

$$\begin{aligned}
&\int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \\
&= \int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y + \mathbb{E}_y y - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \\
&= \int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \\
&\quad + \int\int [\mathbb{E}_y y - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \\
&\quad + \int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y][\mathbb{E}_y y - y] \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x}
\end{aligned}$$

Again, the last term is zero

$$\int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y][\mathbb{E}_y y - y] \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} = \int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y] \left\{ \int [\mathbb{E}_y y - y] p(y|\boldsymbol{x}) \; dy \right\} \; p(\boldsymbol{x}) \; d\boldsymbol{x}$$

$$\begin{aligned}
\int [\mathbb{E}_y y - y] p(y|\boldsymbol{x}) \; dy &= \int \mathbb{E}_y y p(y|\boldsymbol{x}) \; dy - \int y p(y|\boldsymbol{x}) \; dy \\
&= \mathbb{E}_y y \int p(y|\boldsymbol{x}) \; dy - \int y p(y|\boldsymbol{x}) \; dy \\
&= \mathbb{E}_y y - \mathbb{E}_y y \\
&= 0
\end{aligned}$$

And the first term simplifies by marginalizing y.

$$\begin{aligned}
&\int\int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \\
&= \int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y]^2 \; p(\boldsymbol{x}) \; d\boldsymbol{x}
\end{aligned}$$

Finally, we are left with

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}} R(f_{\mathcal{D}}) &= \int\int\int [f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x})]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D} \\
&\quad + \int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y]^2 \; p(\boldsymbol{x}) \; d\boldsymbol{x} \\
&\quad + \int\int [\mathbb{E}_y y - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \\
\mathbb{E}_{\mathcal{D}} R(f_{\mathcal{D}}) &= \text{variance} + \text{bias}^2 + \text{noise}
\end{aligned}$$

This first term is known as the *variance* of the model.

$$\text{variance} = \int\int\int [f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x})]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x} \; P(\mathcal{D}) \; d\mathcal{D}$$

There are two ways to reduce variance:
1) Use a lot of data. Increasing $\mathcal{D}$ will decrease $f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x})$.
2) Use a simple model, $f(\cdot)$, so that $f_{\mathcal{D}}(\cdot)$ does not vary much across datasets.

The second term is known as the *bias* of the model.

$$\text{bias}^2 = \int [\mathbb{E}_{\mathcal{D}} f_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_y y]^2 \; p(\boldsymbol{x}) \; d\boldsymbol{x}$$

We can reduce the bias by using more complex models allowing $f(\cdot)$ to be as flexible as possible to better approximate $\mathbb{E}_y y$. However, this causes the variance to increase.

The third term is known as the *noise* of the mode.

$$\text{noise} = \int \int [\mathbb{E}_y y - y]^2 \; p(\boldsymbol{x}, y) \; dy \; d\boldsymbol{x}$$

There is nothing we can do about noise. Choosing $f(\cdot)$ or $\mathcal{D}$ will not affect it.

As you can see, expected risk (error) breaks down into three terms. Bias and variance both contribute to this error and fight each other over model complexity/simplicity. Increasing the amount of training data will help with the variance contribution to error. The noise is simply inherent and nothing can be done about this.

15 Model Selection

Given some model, $\mathcal{M}$, the Bayesian information criterion (BIC) is used to approximate

$$\log p(\mathcal{D}|\mathcal{M}) \approx \log p(\mathcal{D}|\boldsymbol{w}^{MLE}) - \frac{D}{2} \log N$$

Where $\boldsymbol{w}^{MLE}$ is the maximum likelihood estimation solution to the parameters of the model. For linear regression, we have

$$\text{BIC} \; = -\frac{N}{2} \left(\frac{1}{N} \sum_n (y_n - \boldsymbol{w}^{MLE^{\mathrm{T}}} \boldsymbol{x}_n)^2 \right) - \frac{D}{2} \log N$$

Maximizing the BIC should give us a decent measure of whether our model is too complicated or not complicated enough.

16 Hyperparameter Selection

When training a model and comparing it against other models, it is crucial to separate the training data from the test data. Training your model on the data you test with will cause your model to overfit the data and you have shown absolutely nothing.

When choosing hyperparameters for your model, it is crucial that you tune these hyperparameters with data separate from the training data. Well, there are two ways of going about this.

1) Use training, cross-validation, and test data sets. None of these data sets should overlap. Vary your hyperparameters and train your model with the training dataset. Find the set of hyperparameters that give you the minimal prediction error on the cross-validation set. Then run your prediction on the test dataset and report that as the accuracy of your model.

2) Use training and test datasets that do not overlap and do a N-fold cross-validation in your training data to tune your hyperparameters. What this means is split your training data evenly into N groups. Train your model on N-1 groups and use the left over group as the cross-validation set. Do this N times with the same hyperparameters, iterating over the groups so that each group takes a turn as the cross-validation set. Then average these accuracies. This is the cross-validation accuracy for one set of hyperparameters. Find an optimal set of hyperparameters, then run your prediction on the test dataset and report that as the accuracy of your model.