Summary of PCML 2014 (in progress)

Mohammad Emtiyaz Khan
emtiyaz@gmail.com
EPFL

January 10, 2015

In this note, I will summarize the lectures. I will give a brief overview of each lecture clearly outlining the purpose of that lecture and how does it connect to the core concepts that students need to learn in this course. I will not give any details but if you send me an email about some details missing in the lecture notes, I will update them accordingly. I will keep updating this document if it proves to be useful to students.

Please send corrections to emtiyaz@gmail.com.

Contents

1 Summary of this summary

In machine learning (ML), we are interested in making predictions and decisions based on the *data*. Given a data, we use a *representation* or a *model* that might describe our data well. Usually, a model has *model parameters* which needs to be *learned* to find a good *model-fit* to the data. For learning, we define a *cost function* and use an *algorithm* to minimize it. Most of the time, we can view this process as an optimization problem and/or as a probabilistic modeling problem. Both of these views play important roles in designing new models and algorithms and also in understanding how they relate to each other.

On the practical side, we have to implement, debug, test, and tune machine learning algorithms. However, the most important task is to *generalize* well to unseen data example. For example, *overfitting* is one of the biggest reason for bad performance of ML methods in practice. Theoretically, the generalization error consists of bias and variance terms and we can choose to trade-off one for the other to get good performance. Methods such as *cross-validation*, *regularization*, *model averaging*, *dimensionality reduction* etc. allow us to improve generalization of ML methods.

For this purpose, the two properties of an Ml method are important: statistical and computational. The statistical property is concerned with the question: do I have enough data to learn accurately? On the other hand, the computational property is concerned with the question: do we have enough time to learn accurately? These questions can be answered using tools such as *convexity, computational complexity, identifiability, consistency, optimality* etc.

The first part of the course outlined in Section 2 aims to demonstrate the application of these principles on a regression problem. The first part also lays the foundation of machine learning. In the second part in Section 3, we apply the principles learned in the first part to many other regression and classification problem. In the third part in Section 4, we turn to the problem of unsupervised learning and use the same principles to learn about methods such as mixture models, factor models, and time-series model.

2 Part I: Regression

The first part of the course outlined in Section 2 aims to demonstrate the application of these principles on a regression problem. The first part also lays the foundation of machine learning. To a beginner, this part might seem simplistic to be useful, perhaps because we only deal with linear model. However, principles introduced in this part are absolutely essential to learn any other advanced machine learning methods.

Here is a summary of what we learn in this part.

1. Introduce the regression problem and dataset.
2. Learn how to formulate regression as an optimization problem using cost functions.
3. Understand the statistical-computational trade-off using two concepts: convexity and outliers.
4. Learn about learning based on gradient descent and least squares.
5. Relate optimization formulation to a probabilistic formulation.
6. Fighting overfitting with regularization and cross-validation.
7. Theoretically understand the mechanism behind overfitting using bias-variance trade-off.
8. Apply linear regression to real datasets.

2.1 Introduction to regression

In regression, we are interested in predicting an output y_n given the inputs $\mathbf{x}_n$. We are given a training set which consists of N example pairs $(y_n, \mathbf{x}_n)$. Our goal is to learn a functional relationship $y_n \approx f(\mathbf{x})$ so that we can predict an unseen inputs $\mathbf{x}_*$.

We saw several examples in the lecture, e.g. relationship between height and weight, people's voting behaviours, and predicting sales from advertising in TV, radio, and newspaper. We also gave details about the regression datasets, and defined types of input variables that might occur in practice, e.g. binary, categorical, real etc.

Finally, we discussed two important tasks for regression: prediction and interpretation. In practice, it is not always clear whether a task is prediction or interpretation, but the kind of analysis one would do would differ depending on the task. In prediction task, we treat our ML algorithm as a black box, while for interpretation, we are interested in understanding the relationship between inputs and the output. We covered few real examples to distinguish between the two tasks.

We concluded with the remark that correlation is not causation and that a regression analysis is highly unlikely to also establish a causal relationship.

2.2 Linear regression and cost functions

We started with the simplest type of regression model called linear regression. In linear regression, we assume that the input and the output are related linearly to each other: $y_n \approx \boldsymbol{\beta}^T \widetilde{\mathbf{x}}_n$ (we defined this notation explicitly in the lecture notes). This was our first example of a *model.* The vector $\boldsymbol{\beta}$ is usually referred to as the parameter vector.

Learning involves estimating $\boldsymbol{\beta}$ given a training dataset. To understand this better, we started with two simplest models (see Eq. 4 and 5). The first model is in fact something that is really useful in practice. It amounts to simply estimating the mean of the outputs without using any inputs.

To be able to estimate a "good" parameter, we need to define how costly our mistakes are. We use cost functions to define this. The first cost function that we learned was mean-square error (MSE), see Eq. 6. We did a simple example to show that the parameter can be obtained by minimizing the cost function.

How do we know if a cost function is a good one for the problem in hand? We explored two types of goodness: computational and statistical. We introduced the notion of convexity. Convexity is computationally convenient because when a cost function is convex, it has a unique global minima while no local minimum. In addition, the shape of the function is such that we can find an algorithm which will find this global minimum in a descent time (we will see this in gradient descent lecture). Another advantage of convexity is that sum of convex functions is also convex. Therefore, if we choose a cost function which is convex for one regression pair, its sum over all the data examples is also convex.

Another good characteristics of a cost function is its statistical property. We talked about robustness to outliers. An outlier is an example that is far from most of the examples. In practice, it is difficult to find if a data point is an outlier, but depending on your task, once can choose to call a data example an outlier. We included these examples in the data for project I.

To illustrate these two characteristics on the data, we studied four types of cost functions: mean-square error (MSE), mean-absolute error (MAE), Huber loss, Tukey's bisquare loss. We show that MSE is convex but is sensitive to outliers, while bisquare loss is not convex but is robust to outliers. MAE is convex but not strictly convex and differentiable, and is also reasonably robust to outliers. Huber loss is same as MAE but is differentiable. Some of these loss functions involve setting few parameters which are difficult to set in practice and might involve extra work computationally.

Overall, we see that there is no free lunch (and this is called "no free lunch theorem"). We will continue to see this computational and statistical trade-off throughout the course.

2.3 Linear regression and gradient descent

Given a model and an associated cost function, we can *learn* the parameters, also known as *model fitting, parameter estimation* or simply *learning.* To fit a model, we use an algorithm. In this lecture, we learn about our first algorithm known as gradient descent.

A well-defined cost function has *local* and *global* minimum. These correspond to parameter values that perform better than their neighbourhoods. The simplest way to find these minimum is to exhaustively search the space i.e. the grid-search method. This method does not scale well computationally with the dimensionality of the data and is also sensitive to the *grid size* used.

Gradient descent is a better algorithm and it uses the continuity of a cost function. It makes use of the fact that directions opposite to the gradient might lead us to a local minimum. This is applicable as long as the cost function is differentiable. In *batch* gradient descent, we take a step in the opposite direction to that of the gradient. Value of the step-size is crucial for convergence of gradient descent. Too large a step-size may lead to a divergent algorithm, while too small a value might slow down learning. The advantage of gradient descent, however, is that the computation cost is linear in N and D. For large N and D, even this might be too costly and in that case we can use stochastic gradient descent which computes a noisy estimate of the gradient by subsampling data examples.

Help with revision

1. Revise computational complexity (also see the Wikipedia link in Page 6 of lecture notes).
2. Derive computational complexity of grid-search and gradient descent.
3. Derive gradients for MSE and MAE cost functions.
4. Derive convergence of gradient descent for 1 parameter model (page 12).
5. Implement gradient descent and gain experience in setting of step-size.
6. Learn to assess convergence of gradient descent.
7. Revise linear algebra to understand positive-definite matrices.

2.4 Linear regression and least squares

Gradient descent is an iterative approach and can be applied to many differentiable cost functions. For MSE cost function, it is possible to compute the global minimum in closed form. This is referred to as normal equation. To derive normal equation, we make use of the optimality conditions which state that, at a local minimum, the gradient is zero and Hessian is positive-definite. Taking derivative of MSE cost function and setting it to zero, gives us the normal equation (Eq. 2 in the report). The geometrical interpretation of normal equation is very insightful and will prove to be useful for many other concepts to be learned during the course. Normal equation imply that each data dimension (columns of $\widetilde{\mathbf{X}}$) is perpendicular (or orthogonal) to prediction-error vector.

$$\widetilde{\mathbf{X}}^T(\widetilde{\mathbf{X}}\boldsymbol{\beta}^* - \mathbf{y}) = 0 \tag{1}$$

If $\widetilde{\mathbf{X}}$ is full column rank, we can recover such $\boldsymbol{\beta}^*$ by solving a linear equation (Eq. 4). In practice, $\widetilde{\mathbf{X}}$ may not always be full rank and, even when it is, it might be ill-conditioned. Therefore, a numerically efficient implementation is required. The convenience of least-squares estimation comes at an increased cost of computation. The computation cost is not linear in D anymore.

Help with revision

1. Revise linear algebra to understand why $\widetilde{\mathbf{X}}$ has full rank. Read the Wikipedia page on rank of a matrix.
2. For details on geometrical interpretation, see Bishop 3.1.2. However, better to read this after basis-function expansion. There are some notational differences.
3. Understand robust implementation (see this week's exercise). See Kevin Murphy's section 7.5.2 for details.
4. Understand ill-conditioning. Reading about condition number in Wikipedia will help. Here is another link provided by Dana Kianfar `http://www.cs.uleth.ca/~holzmann/notes/illconditioned.pdf`. We can also use SVD for this purpose.
5. Work out the computational complexity of least-squares (use the Wikipedia page given in gradient descent lecture).

2.5 Ridge regression and overfitting

Linear regression assumes inputs to be linearly related to the output. This may not always be adequate. Fortunately, we can easily extend the linear model to a nonlinear model by simply transforming the input using nonlinear basis-functions i.e. a function such as $\phi_j(\mathbf{x}_n)$. This gives us a new linear regression model with input matrix $\widetilde{\mathbf{\Phi}}$ of size $N \times M$ where M is the number of basis-functions ϕ_j.

A nonlinear model such as this is more expressive than a linear model, but it comes with additional problems of its own. First problem is an easier one and is associated with the ill-conditioning of $\widetilde{\mathbf{\Phi}}$. The basis functions are not always orthogonal (e.g. polynomial basis) and the matrix is usually ill-conditioned.

A more serious problem occurs due to the flexibility of the model to fit the data - a flexible model fit signal well but can also fit noise too. Since we always have finite data in hand, it is difficult to differentiate the signal from the noise and therefore difficult to know whether our model will work well in practice. This phenomenon is also known as Occam's razor which states that in absence of certainty simpler models are better. This problem known as overfitting makes the use of machine learning methods essential.

Both of these problems are solved using regularization. We add a *penalization* term to the cost function to control for model complexity. Usually, an L_2 term is employed (Eq. 5) and is weighted by a positive scalar term λ which controls the extent of penalization. Throughout the course we will witness the use of L_2 regularizer for many models. We will also see different ways of regularization when we study SVM, Bayesian methods such as Gaussian processes, time-series models, and graphical models.

Since we do not know the ratio of the signal and the noise in our data, setting λ is tricky in practice. Cross-validation (CV) solves this problem by simulating the reality so that we can minimize the mistakes made on future unseen data. In K-fold CV, we split the data into multiple disjoint K sets and repeat learning on $K - 1$ subsets while testing on the leftover subset. Depending on the amount of the data available, this might give us an unbiased estimate of error on unseen data. We can use use this error to set λ.

Help with revision

1. You can find more details on basis-function models in Section 3.1 of Bishop (3.1.1 might be confusing and can be skipped).
2. Details on ridge regression in Section 3.1.4 of Bishop.
3. Implement ridge regression and understand how it solves the two problems.
4. Implement cross-validation and gain experience on setting λ and K.
5. Details on unbiasedness of cross-validation is in Section 7.10 of HTF. There is also a discussion on this in the forum.
6. Read the section on overfitting in the paper by Pedro Domingos (section 3 and 5).

2.6 Linear regression and bias-variance decomposition

In prediction, our goal is to *generalize* well on unseen data. Sometimes we might observe bad performance since we used a simple model, but bad performance might also arise due to overfitting caused by a complicated model. Methods such as cross-validation give us a tool to mimic the reality and trade-off model complexity for performance on unseen data. In this lecture, we formalize the theory behind this.

We can decompose the (expected) test and train error into terms containing bias and variance. Our fit might be biased because we chose a simplistic model or because we did not have enough data or appropriate method of estimation. First kind of bias is called model bias and occurs since our model might be too simplistic for the data in hand, e.g. a linear model for nonlinear data. Second kind of bias is called estimation bias and occurs mainly due to finite amount of data, but may also occur due to inappropriate method of estimation e.g. choosing a suboptimal solution. High variance exists due to a complex model which gives completely different test error for different splits of the data. Both bias and variance contribute to test error. Increasing model complexity increases the variance of your estimator while reducing the bias.

We derive the bias-variance decomposition for the nonlinear regression model from previous lecture and show that ridge regression (or regularization) increases the estimation bias while reducing the variance of the estimator. Note that for least-squares estimation as $N \to \infty$, estimation bias goes to zero, but model bias might still exist. In the finite data case, however, the variance reduction obtained using ridge regression usually helps to improve the performance even though the model bias is increased.

Help with revision

1. Clearly understand the definition of expected test and train errors (do the exercise).
2. Details of the derivation are in Section 7.2 and 7.3 of HTF (not easy to read, takes time).
3. Learn to derive bias-variance decomposition.
4. Visualize bias-variance decomposition (see the exercise).
5. Figure in page 9 is most useful to understand what was taught in the lecture.

2.7 Linear regression and maximum likelihood

In previous lectures, we saw many models and algorithms for regression. In this lecture, we will learn about a probabilistic point of view for them. This view will help us understand when and why these methods are suitable for the problem in hand. Most importantly, this view will let us generalize our learning to many more complicated models.

We show that minimizing mean-square error is equivalent to maximizing the likelihood of the data after assuming that the data comes from a Gaussian distribution.

$$\arg\min_{\beta} \mathcal{L}_{mse}(\boldsymbol{\beta}) = \arg\max_{\beta} \mathcal{L}_{lik}(\boldsymbol{\beta}) \tag{2}$$

This might seem trivial at first, but this view helps us to prove many important properties of our estimators. In the lecture, we show that least-squares is consistent i.e. it converge to the true value as we increase the data, and that it is asymptotically normal i.e. its distribution converges to a Gaussian. Not only this, we also show that least-squares estimate is optimal (under the assumption that the Gaussian model is the correct model i.e. no model bias).

The probabilistic point of view is most useful in deriving new cost functions. In fact, most cost functions can be obtained using maximum-likelihood approach on a probabilistic model. We will use this approach almost in all the models that we will study in this course.

Help with revision

1. Understand the big picture: why is probabilistic view useful?
2. Understand the notation $p(\mathbf{x}|\boldsymbol{\theta})$.
3. Revise Gaussian distribution (you must know the formula well).
4. Clearly understand the relationship between MSE and log-likelihood for least-squares.
5. Read Wikipedia page for definition of consistency.
6. Section 2.6 derives the MAE cost function using a Laplace distribution.

2.8 Testing linear regression

RMSE and cross-validation are not the only way to test performance of regression model. We did not have time to cover other methods in the lectures, but an understanding of these methods helps us to apply what we learned in the course. Details of these methods can be found in pages 47-49 of HTF book. An easier version is available in Chapter 3 of JWHT book.

3 Part II: More Regression and Classification Methods

The second part of the course applies the principles learned in the first part to derive more regression and classification methods. We start with logistic regression and derive the cost function using the maximum likelihood framework. We learn about algorithms such as Newton's method and IRLS. This serves as an introductory example of a general class of models called generalized linear model (GLM) where the idea is to map the output of linear regression to a non-Gaussian output (e.g. Bernoulli or Poisson distribution).

Next, we discuss a very important aspect of machine learning: flexibility of a model. We introduce k-NN which is a much more flexible method compare to a linear model and show that it may preform badly due to *curse of dimensionality.* In high dimensions, all data points are extremely far from each other making learning difficult. To resolve this issue, we introduce kernel methods which allows us to tune the complexity of a model. We obtain a range of methods using kernel methods. We discuss SVMs for binary classification, and Gaussian processes for regression.

We also discuss artificial neural network which is a different kind of flexible model.

3.1 Binary classification and logistic regression

In binary classification, the output variable y_n can take only binary values. This is in contrast to regression where y_n is real valued. A binary state is also called a class and sometimes outputs are sometimes referred to as labels. The simplest way to deal with binary classification is to simply use linear regression. We can recode the two classes as -1 and +1 respectively and do a least-squares fit. The advantage of this approach is that it is simple to implement and has a closed-form solution (using the normal equation). However, this approach does not ensure that the prediction probabilities are between 0 and 1. Also it is difficult to extend this approach to multi-class problems.

We take a probabilistic approach to derive a better model (or cost function). We extend linear regression to binary outputs by using the logistic function $\sigma(x)$ which maps the real-valued output to the probability of the binary output. We then use the maximum likelihood framework to derive a cost function.

Taking the gradient of this cost function give rise to an equation similar to the normal equation.

$$\widetilde{\mathbf{X}}^T [\sigma(\widetilde{\mathbf{X}}\boldsymbol{\beta}^*) - \mathbf{y}] = 0 \tag{3}$$

This is not a coincidence. In fact, this is an instance of a larger class of models called generalized linear model (GLM). In these models, a linear regression output is mapped to the probability of the output, e.g. to obtain the probability of count outputs we can use a Poisson distribution.

Cost functions obtained in such a way are usually computationally convenient e.g. they are convex and sometimes even strictly convex (see page 4 in part II of notes). This holds for the whole GLM class. We can obtain faster convergent algorithms using Newton's method which can also be interpreted (and implemented) as a sequence of least-squares problems thereby allowing easy implementation (see IRLS in page 7 of part II). In addition, all statistical properties of least-squares are preserved.

A specific problem with logistic regression is that the MLE cost function can be unbounded when the data is linearly separable (i.e. there is a linear plane that separates both classes perfectly with zero mistakes on the training data). To remove this problem, we can add an L_2 regularizer that makes the cost function bounded. This is also known penalized logistic regression and is equivalent to ridge regression for least-squares. This penalization term not only improves the ill-conditioning and ill-posedness of the problem, but also helps to reduce overfitting (similar to least-squares case).

Logistic regression is related to perceptron algorithms and also neural networks. Extension to multiclass classification can be easily done using a softmax link function, which is again an instance of generalized linear model.

Help with revision

1. Understand the logistic function and its link to GLM (see Wikipedia page on GLM).
2. Learn to derive the cost function using maximum likelihood estimation.
3. Understand the normal equation (Eq. 12 in part I).
4. Understand the interpretation of log-odds (see JWHT Chapter 3).
5. Learn to prove convexity using the positive-definite property of the Hessian.
6. Implement Newton's method (see the exercise).
7. Understand the relationship with IRLS (page 7).

3.2 Curse of dimensionality and k-NN

We revisit the classification problem and introduce k-nearest neighbour (kNN) classifier. The idea is extremely simple: a given point is classified to the class which contains majority of its k nearest neighbours. This doesn't even require any training. However, the decision boundaries are very sensitive to the choice of k (see Fig. 2.1 and 2.2). Obviously, choice of k affects the bias and variance and we can set its value using cross validation. An example is shown in Fig. 2.4 where it appears that k-NN might do much better than linear regression. This is also indicated in the fact that the decision boundaries obtained using k-NN are non-linear and therefore are much more flexible than a linear method.

Unfortunately, this flexibility might not help when our data is high dimensional. This is referred to as the curse of dimensionality. The main reason behind the curse is that, with the dimensionality increase, every data point becomes arbitrarily far from every other data point and therefore the choice of nearest neighbour becomes random. In high dimension, data only covers a tiny fraction of the input space, making generalization extremely difficult. This is very counter-intuitive and Pedro Domingo puts it in the right words "our intuitions fail in high dimensions".

In high dimension, both bias and the variance of a method increase. This is true for almost all methods. The rate of increase is really high for methods such as k-NN that directly depend on the distance between data examples. We can slow down this rate by using a simpler model that makes more model assumptions. For example, for linear regression, variance grows linearly with dimensionality which is much lower than an exponential rate for k-NN. In the next few lectures, we explore a whole spectrum of models between the linear model and k-NN. These models makes different kind of assumptions and may might perform better.

An important point to remember is that you might think that gathering more input variables never hurts, since at the worst they provide no new information about the output. But in fact their benefits may be out-weighted by the curse of dimensionality.

Bottom line : reduce your feature dimensionality whenever possible.

Help with revision

1. The material in the lecture notes is taken from HTF Section 2.3.3 and 2.4. Make sure to work the examples of Figure 2.6. The description in the lecture notes is simplified, so you can refer to lecture notes to understand the contents of the book.

2. It is very useful to read Pedro Domingos insights on curse of dimensionality (see Section 6 of his paper).

3.3 Kernel methods and ridge regression

k-NN makes use of the distances (or similarities) between data examples for prediction. The idea of distance can be generalized using kernels which essentially define distance metrics on a feature space. With the use of a kernel, a regression or classification method can then be *kernelized*.

Kernelization is a natural consequence of the representer theorem which states that for every $\boldsymbol{\beta}^*$ which minimizes the cost function below, we can find $\boldsymbol{\beta}^* = \mathbf{X}^T\boldsymbol{\alpha}^*$ ($\mathcal{L}$ is any cost function).

$$\min_{\beta} \tfrac{1}{2}\sum_{n=1}^{N}\mathcal{L}(y_n, \widetilde{\mathbf{x}}_n^T\boldsymbol{\beta}) + \tfrac{1}{2}\lambda\sum_{j=1}^{D}\beta_j^2 \tag{4}$$

Note that $\boldsymbol{\beta}^*$ is in $\mathbb{R}^D$ while $\boldsymbol{\alpha}^*$ is in $\mathbb{R}^N$. The representer theorem tells that the two spaces are connected through $\mathbf{X}$ and that we can always obtain $\boldsymbol{\beta}^*$ given $\boldsymbol{\alpha}^*$ and vice-versa (if $\mathbf{X}$ is rectangular we can use SVD for inversion). We can also write an equivalent optimization problem wrt $\boldsymbol{\alpha}$ to obtain $\boldsymbol{\alpha}^*$.

When $\mathcal{L}$ is convex, we can do this transformation using convex duality which essentially converts a *minimization* problem over $\boldsymbol{\beta}$ to a *maximization* problem over $\boldsymbol{\alpha}$. The first problem is called primal while the latter is called the dual. There are several advantages to work with the dual instead of the primal. The dual is usually computationally efficient (although not always). More importantly, the dual is expressed in terms of the kernel matrix $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$.

The advantages of using a kernel are numerous. First of all, we do not have to explicitly specify the feature transformation $\boldsymbol{\phi}(\mathbf{x})$ (see page 9 of the lecture notes). Computationally it might be cheaper to work with the kernel matrix than the original $\boldsymbol{\Phi}$ matrix (an example is given in page 9). Second, we can define feature transformations that are of infinite length! This is possible since the kernel matrix is still finite dimensional. Finally, a variety of kernels are available and can be chosen for a specific application. We can also design new kernels by taking composition of existing kernels.

But how do we know that a matrix K is a kernel? It is simple: if $\mathbf{K}$ is symmetric positive-semi-definite, it is a kernel.

Help with revision

1. Clearly understand the relationship $\boldsymbol{\beta}^* = \mathbf{X}^T\boldsymbol{\alpha}^*$. See page 4 in the lecture notes. Understand the statement for the representer theorem.

2. Prove that for ridge regression primal and dual give the same solution (page 6).

3. Get familiar with various examples of kernels. See page 9 of the lecture notes and Section 6.2 of Bishop on examples of kernel construction. Read Kevin's book Section 14.2 for examples of kernels.

4. Revise and understand the difference between positive-definite and positive-semi-definite matrices.

5. If curious about infinite $\boldsymbol{\phi}$, see Matthias Seeger's notes (uploaded in Moodle).

3.4 Kernel methods and SVMs

Kernel methods play an important role for support vector machines (SVMs). SVMs optimize a cost function that is very similar to logistic regression. The difference is that the logistic loss is replaced by a hinge loss. This leads to a sparse solution which makes SVM fast to train. Use of hinge loss also facilitates the maximization of margin. A large margin gives rise to a stable method that does not change much with small changes in the data (see the figure in page 5 for an intuitive explanation, we did not cover this in the class since the details are involved but if you are interested you can read the corresponding chapter in HTF).

The difficulty with the optimization of hinge loss is that it is not differentiable. However, if we dualize the problem and optimize the dual with respect to $\boldsymbol{\alpha}$, then the problem has a very convenient form. First, the optimization is simply a constrained least-squares problem which can be solved efficiently. Second, the problem is naturally kernelized i.e. it is written in terms of $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$.

The most important consequence, however, is that the solution $\boldsymbol{\alpha}^*$ is sparse. It turns out that α_n is non-zero only for the data example n that lie inside the margin. These data examples are called support vectors since they support the decision boundary. Sparsity is the key feature that makes SVMs fast to train which is perhaps one of the main reason for popularity of SVMs (and other maximum margin methods).

Help with revision

1. Understand hinge loss and the margin (page 4 and 5).
2. Visualization hinge loss (see Notes page in the lecture notes).
3. Get comfortable with the duality principle (page 7).
4. Work out SVM derivation.
5. Clearly understand the reasons why dual is better than the primal (page 11).
6. What does support vector mean? Why do they arise? Where do they lie in the data space? Page 12 and 13.

3.5 Bayesian methods and Gaussian processes

Methods we have seen so far do not (naturally) produce an uncertainty estimate. Gaussian processes regression is a *non-parametric* method that compute a probability distribution over predictions. Methods based on Gaussian processes (GP) are non-parametric since they do not use a parametric form for modeling, e.g. GP regression does not have any parameters $\boldsymbol{\beta}$ as in other regression models rather it uses latent variables to correlate data examples.

Think of a simple example consisting of two data pairs: $(y_1, \mathbf{x}_1)$ and $(y_2, \mathbf{x}_2)$. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are close by (in some metric space) then we might expect that y_1 and y_2 are correlated, otherwise these outputs should be independent. A GP prior encodes such dependencies. We associate each output y_n with a latent variables f_n. We form a kernel matrix $\mathbf{K}$ using inputs $\mathbf{x}_n$ and use it as the prior covariance of a Gaussian distribution. This is a Gaussian *process* prior since the latent functions $\mathbf{f}$ defined this way are random *processes* rather than just random variables (this point is subtle and plays an important role in understanding why GPs are useful modeling tools but a beginner can ignore this fact for now). More importantly, this construction makes sure that if $\mathbf{x}_i$ is close to $\mathbf{x}_j$ then f_i might be correlated with f_j.

The final model is as follows: $\mathbf{f}$ follow a Gaussian distribution with covariance $\mathbf{K}$ and then $\mathbf{y}$ is generated using a Gaussian distribution with mean $\mathbf{f}$. Therefore a Gaussian prior with a Gaussian likelihood. How do we find a good value of $\mathbf{f}$? An obvious approach is to optimize MSE cost function but the problem with this approach (among many others) is that this method will not give us an uncertainty measure (at least naturally). It turns out that we can compute the full posterior distribution over $\mathbf{f}$ instead of just computing the maximum likelihood estimate.

To understand this, we establish a connection between least-squares and ridge regression (see page 3-5). Least-squares finds a maximum likelihood estimate under the assumption that the likelihood is Gaussian. We can show that ridge regression is similar but differs in two aspects. First it assumes a Gaussian prior distribution over $\boldsymbol{\beta}$, and second, it finds maximum of the posterior $p(\boldsymbol{\beta}|\mathbf{y}, \mathbf{X})$ instead of the likelihood. It turns out that computing the maximum is not much more cheaper than computing the full posterior distribution. The reason is that when we have Gaussian prior and a Gaussian likelihood, our posterior is also Gaussian (this is also a result of something called the *conjugate prior* which we study later).

This result directly applies to GP regression. We have a Gaussian process prior over $\mathbf{f}$ with a Gaussian likelihood for $\mathbf{y}$. Hence the posterior $p(\mathbf{f}|\mathbf{y}, \mathbf{X})$ is also going to be Gaussian! To compute the expression (in page 1), we can use the closed-form expression given in Bishop's book chapter 2.

A major issue with GPs is computational. You can see that computing predictions require inverting the covariance matrix which is of size $N \times N$ and could cost $O(N^3)$ in the worst case. Sparse approximations are usually employed to reduce the computation.

In general, the method of computing a posterior distribution is related to Bayesian methods. Bayesian methods marginalize out the irrelevant variable (called *nuisance* variables) giving rise to a new cost function that only depends on the variables of interest. Hence Bayesian methods are another way of finding robust cost functions. We will see a few more examples of Bayesian methods in this course and this point is worth remembering for those examples.

Help with revision

1. Derive Eq. 6, 8, and 11 in the lecture notes to understand MAP estimation. You can find additional explanations in Bishop Section 6.4.1 and 6.4.2 about this.

2. Use above equations to understand Eq. 15.

3. Derive Eq. 3 using Bayes rule and using the Gaussian formula (Eq. 2.113-2.117 in Bishop's book).

4. Understand Bayesian averaging (marginalization) introduced in page 8 (this may not "click" for beginners, but you can revisit this after you have seen few other Bayesian methods).

3.6 Artificial neural networks

Artificial neural networks employs several nonlinear layers containing several linear models with outputs of each linear model transformed using a nonlinear function. Simplest version is perhaps a 1 layer neural network with 1 binary output with a sigmoid transformation. This is a logistic regression model.

Outputs of each linear model are called *activations*. We denote them by $a_{nm}^{(k)}$ for m'th linear model in k'th layer of n'th data point. The activations are transformed using a non-linear function $h(\cdot)$ to get hidden units, denoted by $z_{nm}^{(k)}$. We combine hidden units of the last layer to get (one or many) outputs. It is usually convenient to write this using matrix-vector product (page 8).

Each linear model has parameters associated with it. Denote the parameter m'th linear model in the k'th layer by $\boldsymbol{\beta}_m^{(k)}$. If there are M hidden units in that layer, then there will be M such parameter vectors. We can form a matrix $\mathbf{B}^{(k)}$ that contains these parameters as rows. We need to learn K such matrices for a K-layer neural network.

For learning, we can use a MSE cost function. If the output is binary, we can employ a logistic loss (log-loss) function. If there are multiple outputs we minimize the sum of the cost functions for all outputs. This part is trivial. Learning, however, can be tricky because the model is unidentifiable (see page 12) and obviously contains numerous local minima. The most popular method is to use stochastic gradient descent for learning.

Another difficulty lies in computing the gradient (for gradient descent). Due to cascading, all layers depend on each other making gradient computation a tedious task. Backpropagation solves this problem. The key point here is that the gradient with respect to $\mathbf{B}^{(k)}$ can be written in terms of gradients with respect to $\mathbf{a}_{nm}^{(k)}$ and then the gradients wrt to activations of k'th layer can be expressed in terms of gradients wrt of $(k+1)$'th layer. So we first do a forward pass to compute all activations. Afterwards, we compute the gradient wrt the activations of the last layer and then go backwards computing the gradient of the second last layer, and so on.

Neural networks are universal density approximators, meaning that they can approximate any function arbitrary well. Consequently there is a big danger of overfitting. A standard way to combat overfitting is to add an L_2 regularizer, however one has to be careful and use different regularization coefficients for different layers. Another way to regularize is to use *early-stopping.*

If trained properly neural networks can perform extremely well. The power of neural networks lies in the fact that the hidden units learn a distributed representation of of the data. Each output is predicted with a combination of the hidden units therefore borrowing statistical strength from multiple representations (similar to ensemble methods such as random forests). The multiple layers can capture different details about the data and can at times learn meaningful representations. A great deal of work for many years has gone into training of neural networks and many recent breakthroughs have brought us to the era of *deep learning.*

For all this to work, a well structured neural network is essentially important. One can think of this as a form of regularization. In this regard, models such as convolutional neural networks have turned out to be very useful. There is no doubt that neural networks will continue to be extremely useful as we learn more about how they work and how we can train them better.

Help with revision

1. Understand the notation and also the structure of neural networks (page 5-7). Clearly understand the definition of activations and hidden units, as well as extension to multiple outputs.

2. Simulated example in page 9 gives an insight about the distributed representation of neural networks. Details are in Bishop's book.

3. Understand identifiability issues. Details are in Bishop Section 5.1.1.

4. Understand back-propagation. Work out a numerical example to get familiar with Eq. 11-13.

5. Read history of neural networks from Kevin Murphy's book Section 16.5.3.

4 Part III : Unsupervised Learning

Unsupervised leaning aims to find patterns in the data that reflect statistical structures present in the data. This form of learning is very useful and widely applicable since in most applications labels (or outputs) are unavailable. It is also believed that unsupervised learning is more common in the brain compared to supervised learning. Unsupervised learning therefore might play an important role in the design of machines that mimic the brain.

In this course, we explore three types of unsupervised learning approaches: mixture models, factor models, and graphical models.

Help with revision

1. Read Peter Dayan's paper on unsupervised learning.

4.1 Mixture models and K-means

Clustering is a form of unsupervised learning where we group many data vectors $\mathbf{x}_n$ into separate clusters. *Mixture models* define the clustering using a few *prototype* vectors $\boldsymbol{\mu}_k$ (for k equals 1 to K) and assigning data vectors to them. The assignments and prototypes are chosen to minimize sum of mean-square distances between prototypes and the data vectors assigned to them (see Eq. 1). At first it appears that the optimization might be difficult, but K-means algorithm simplifies it by iterating between computing prototypes given assignments and then computing assignments given prototypes. This is a form of *coordinate ascent* algorithm and the famous EM algorithm also falls under the same category.

K-means cost can also be written as a probabilistic model under which data vectors are independently distributed and follow a Gaussian distribution with the mean equal to the prototype they are assigned to and the covariance equal to the identity matrix.

A problem with K-means is that it only captures spherical Gaussian clusters (not elliptical). Another issue is that each data vector is assigned to only one cluster. These *hard assignments* may not be a good idea in practice (remember project 1?). Most of the time, it is better to have *soft* assignments. We solve these problems using GMM in the next lecture.

Help with revision

1. Understand the iterative algorithm for K-means. Why is this problem difficult to optimize and how iterative algorithm makes it simpler?
2. Understand the probabilistic model behind K-means cost function (page 15).

4.2 Gaussian mixture model and expectation-maximization

The two problems discussed in the K-means lecture can be solved using GMM. The first problem can be solved by allowing non-identity covariance matrices for Gaussian clusters. The second problem can be solved by making assignments to be random variables rather than parameters. We assume assignments to follow the multinomial distribution. The resulting model is called *Gaussian mixture model* since the *marginal distribution* over $\mathbf{x}_n$, obtained by marginalizing over assignments, is a mixture of Gaussians (Eq. 4).

Since assignments are random variables, the parameters of the model consists of mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$ for all k (denote by $\boldsymbol{\theta}$). Given $\boldsymbol{\theta}$, we can compute the probability of each data vector's assignment to a cluster (Eq. 5) using Bayes rule. During learning, we have to compute $\boldsymbol{\theta}$ for which we can use a maximum likelihood estimate. This ML estimate is not identifiable and might suffer from *singularity.* In addition, this cost is not convex and we might only find a local minima.

We can apply an iterative method similar to K-means. This method is an example of expectation-maximization (EM) algorithm where we iterate between computing the posterior probability over the assignments given $\boldsymbol{\theta}$ and then estimating $\boldsymbol{\theta}$ given the assignments. This is again a coordinate ascent algorithm, very similar to K-means. We start with an initial parameter estimate $\boldsymbol{\theta}^{(1)}$ in the first iteration. Every iteration consists of two steps. The first step, called the expectation or E-step, computes a lower bound to the cost function given a $\boldsymbol{\theta}^{(k)}$ in the k'th iteration. The lower bound is obtained using Jensen's inequality which is simply using concavity of log (Eq. 9). The second step (called the maximization or M-step) finds the maximum of the lower bound computing $\boldsymbol{\theta}^{(k+1)}$.

EM algorithm is applicable to a general class of latent variance models. Given a model with independent data vectors $\mathbf{x}_n$ each of them associated with latent vector $\mathbf{z}_n$ and parameter $\boldsymbol{\theta}$, we can learn $\boldsymbol{\theta}$ by maximizing $\sum_n \log p(\mathbf{x}_n|\boldsymbol{\theta})$ (the maximum likelihood estimator). In the E-step, we compute the posterior $p(\mathbf{z}_n|\mathbf{x}_n, \boldsymbol{\theta}^{(k)})$ and then use Jensen's inequality to compute a lower bound. Note that it is essential to compute the posterior and this computation is possible when the prior $p(\mathbf{z}_n|\boldsymbol{\theta})$ is a *conjugate prior* to the likelihood $p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\theta})$ (making Bayes rule easier to apply). In the M-step this lower bound is maximized. This step is easy when the lower bound is concave and differentiable. Compactly the two steps can be written as follows:

$$\boldsymbol{\theta}^{(k+1)} = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{p(z_n|x_n,\theta^{(k)})}[\log p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\theta})] \tag{5}$$

How do we choose K? This is a model selection problem, a challenging task. We can use CV which may give suboptimal results. Many advanced methods exist to solve this problem though.

Mixture models can be combined with many other supervised and unsupervised models. For example, we can have mixture of linear regression or mixture of factor models. See Bishop's Chapter 14.5 for details.

Help with revision

1. Understand K-means extension to GMM. Why do we need to treat r_n as a random variable? Identify the joint, likelihood, prior, and marginal distributions respectively. Understand the use of Bayes rule that relates all these distributions together.
2. Learn to compute the posterior distribution.
3. Understand identifiability and difficult with the maximum-likelihood estimation.
4. Understand how EM circumvents this problem. Derive the lower bound using Jensen's inequality. Maximize the lower bound in the M-step (both steps very important for the exam).
5. Understand the relation between EM and K-means iterative algorithm.
6. Relate the lower bound to EM for probabilistic models in general.
7. Read Bishop Section 14.5 to learn about conditional mixture models.

4.3 Factor models: PCA, SVD, and low rank approximations

Latent factor models utilize unobserved (hidden) variables to model correlation in the data. Principal component analysis (PCA) is a popular latent factor model, widely used for dimensionality reduction, compression, feature extraction, and data visualization. PCA in itself is a very old method and has been used in many different forms. In this lecture, we study a few forms of PCA and how they relate to each other.

An important application of PCA is dimensionality reduction. Each vector $\mathbf{x}_n \in \mathbb{R}^D$ is projected to smaller length $\mathbf{z}_n \in \mathbb{R}^M$ such that $M < D$ but $\mathbf{x}_n$ is close to $\mathbf{W}\mathbf{z}_n$. The goal is to learn both $\mathbf{W}$ and $\mathbf{z}_n, \forall n$. A straightforward approach is to choose these to minimize the reconstruction error (similar to MSE). The resulting cost function unfortunately is not jointly convex wrt $\mathbf{W}$ and $\mathbf{Z}$ and also the underlying model is unidentifiable. We can, however, use an alternating algorithm which is guaranteed to converge to a local minima. Fortunately, this algorithm is very efficient where each step amounts to solving a ridge regression problem. The method is also known as alternating least-squares (ALS).

The cost function that we consider (in Eq. 1) is equivalent to assuming a Gaussian distribution on the data, therefore we can interpret the solution as a maximum likelihood estimate. Using the generalized linear model approach, we can easily extend the cost function to non-Gaussian data, e.g. for count data we can use Poisson distribution and for binary data we can use logistic function (similar to logistic regression). We can also, however, transform the data to make it Gaussian (as we did for the song recommendation dataset).

The above method of dimensionality reduction is also referred to as a "low-rank approximation". There are other ways of finding the PCA solution. One such way is to use singular value decomposition (SVD) where we can retain first few singular values to get a low-rank approximation. Such approximation is also justified from a *spectral* view where higher singular values contain the low-frequency information. Using SVD, we can see that PCA is equivalent to a decorrelation in the dataset.

Help with revision

1. Understand the cost function for dimensionality reduction (Eq. 1) and the algorithm to optimize it.
2. Understand the implications of an MSE cost function (page 6). Which probabilistic model does this correspond to? How can you reduce overfitting? When and why should you pre-process the data to make it Gaussian? Use what you learned from Project 2 to answer these questions. Also, work out the sample question on PCA for count data and relate it to these questions.
3. Understand SVD and its relation to low-rank approximation and PCA.

4.4 Bayesian networks and belief propagation

I will not be adding this.