Companion to Machine Learning

Rohan Kumar

Contents

Introduction

0.1 What is Machine Learning

Machine Learning is the field of study that gives computers the ability to learn from data without being explicitly programmed. This is good for problems that require a lot of fine-tuning or for which using a traditional approach yields no good solution. Machine Learning's data dependency allows it to adapt to new data and gain insight for complex problems and large amounts of data.

0.2 Applications of Machine Learning

Machine Learning can be used for a range of tasks and can be seen used in:

- Analyzing images of products on a production line to automatically classify them (Convolutional Neural Net)
- Forecasting company revenue based on performance metrics (Regression or Neural Net)
- Automatically classifying news articles (NLP using Recurrent Neural Networks)
- Summarizing long documents automatically (Natural Language Processing)
- Building intelligent bot for a game (Reinforcement Learning)

0.3 Types of Machine Learning

0.3.1 Supervised Learning

In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels. (e.g determining if an email is spam would be trained a dataset of example emails labelled as spam or not spam.)

Some commonly used supervised learning algorithms are:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural Networks

0.3.2 Unsupervised Learning

In unsupervised learning, the training data is unlabeled and the system tries to learn without guidance. The system will try and automatically draw inferences and conclusions about the data and group it as such. (e.g. having a lot of data about blog visitors. Using a clustering

algorithm we can group and detect similar visitors).

Some important unsupervised learning algorithms are:

- Clustering
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis
- Anomaly detection and novelty detection
  - One-class SVM
  - Isolation Forest
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally Linear Embedding (LLE)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
  - Apriori
  - Eclat

0.3.3 Semisupervised Learning

Labelling can be very time-consuming and costly, often there will be plenty of unlabelled and a few labelled instances. Algorithms that deal with data that is partially labeled is called semi-supervised learning. A good example of this is Google Photos. Google clusters and groups your photos based on facial recognition (unsupervised) and then you can label one photo and it will be able to label every picture like that (supervised). Most semi-supervised learning algorithms are combinations of unsupervised and supervised algorithms.

0.3.4 Reinforcement Learning

Reinforcement Learning is a learning algorithm based on a reward system. The learning system, called an agent, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards). It will then learn by itself what the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

Sources

Throughout this compendium, each piece of information will be formatted as such.

Name / Description of fact Source

Information about fact.

The location which currently contains "Source" could potentially be filled with a variety of sources. Here is how to find the source based off the shortened form.

- **Hands-On Machine Learning** refers to Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition by Aurélien Géron

1 Data Analysis

TODO:

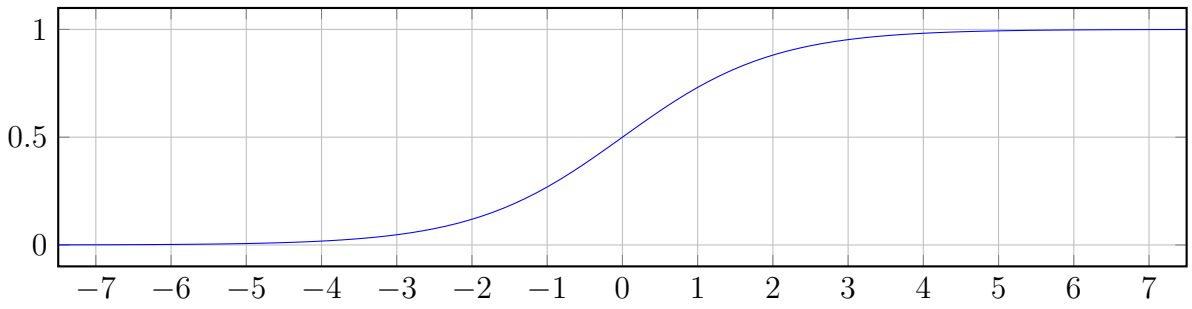
- Notation Sheet
- Data Model
- Overfitting/Underfitting
- Feature Normalization
- Types of Loss Functions
- Show what the gaussian distribution expansion is

2 Activation Functions

2.1 Sigmoid Function

The *sigmoid function* denoted $\sigma(\cdot)$ outputs a number between 0 and 1. It is defined as

$$\sigma(t) = \frac{1}{1 + exp(-t)}$$



The key property of the sigmoid function is that $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$, so a sigmoid function is useful for classification since it can predict 1 when $\boldsymbol{w}^T\boldsymbol{x}$ is positive and 0 if it is negative.

3 Convex Optimization

In machine learning we can often turn problems into convex functions and simplify the problem into finding the global minima of the function, which in essence is minimizing the training error. One of the key theorem's of a convex function is that the local minimum of a convex function is also a global minimum. Therefore we can apply many methods to find parameters that satisfy the global minima.

3.1 Normal Solution

To find the value of our parameter (generally $\boldsymbol{w}$) that minimizes the cost function, there is a closed-form solution. We can express this closed form solution for any convex loss function as follows.

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$$

The limitations of using the Normal Solution is that we usually have to compute the inverse of $X^T X$ which is a $(n+1) \times (n+1)$ matrix (where n is the number of features). The computational complexity of inverting such a matrix is typically about $O(n^{2.4})$ to $O(n^3)$, depending on the implementation. Some of the other approaches below are better suited for cases where there are a large number of features or too many training instances to fit in memory.

3.2 Gradient Descent

Gradient Descent is a generic optimization algorithm capable of finding optimal solution to a wide range of problems. The general ida is to tweak parameters iteratively in order to minimize a cost function. The main concept utilized in gradient descent is to measure the local gradient of the error with regard to the parameter vector and move in the direction of the descending gradient. Once the gradient is zero, you have reached the minimum.

You start with filling your parameter $\boldsymbol{w}$ with random values (random initialization. Then you improve it gradually, taking small steps at a time, each step attempting to decrease the cost function, until the algorithm converges to a minimum. One important parameter of Gradient Descent is the size of the steps, determined by the *learning rate* hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, if the learning rate is too high, you might jump across the minimum possibly, higher than you were before and potentially make the algorithm diverge. One gradient descent technique is having a learning rate that changes as you approach the minimum to prevent overshoot, also called the learning schedule.

A limitation of Gradient Descent is when the cost function we are dealing with is not a convex function. In this case holes, ridges, irregular terrain will make the convergence to the minimum difficult.

3.2.1 Batch Gradient Descent

To implement Gradient Descent, you need to compute the gradient of the cost function with regard to each model parameter $\boldsymbol{w}_j$ - how much the cost function will change if you change $\boldsymbol{w}_j$ a little bit. This is equivalent to the partial derivative of the cost function with regard to the parameter $\boldsymbol{w}_j$. For the entire parameter vector $\boldsymbol{w}$ we can denoted the gradient vector as $\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$.

Once we have the gradient vector, which points uphill, we descend in the opposite direction (subtract $\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$ from $\boldsymbol{w}$). This is where we use our learning rate α to determine the size

of the downhill step.

$$\boldsymbol{w}^{(nextstep)} = \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

The limitation of Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.

3.2.2 Stochastic Gradient Descent

Stochastic Gradient Descent picks a random instance in the training set at every step and computes the gradients based on only that single instance. This makes the algorithm much faster and also makes it possible to train on huge training sets. However, due to its stochastic nature, this algorithm will bounce up and down, decreasing only on average. Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down. Therefore, once the algorithm stops, the final parameter values are good, but not optimal.

This can actually help when the cost function is very irregular (not convex) as it can help the algorithm jump out of a local minima. One solution to the problem of being unable to settle at the minimum is gradually reducing the learning rate. The steps start out large (helps make quick progress and escape local minima), then get smaller and smaller, allowing the algorithm to settle at the global minima. The function that determines the learning rate is called the *learning schedule*.

3.2.3 Mini-batch Gradient Descent

Mini-batch GD is a combination of Batch GD and Stochastic GD. At each step, instead of computing the gradients based on the full training set or based on just one instance, Mini-batch GD computes the gradient on small random sets of instances called *mini-batches*. The main advantage of this over Stochastic GD is that you get a performance boost from hardware optimization of matrix operations. Mini-batch will perform better to get closer to the minimum than Stochastic GD but it may be harder for it to escape local minima.

4 Instance-Based Learning

4.1 Parametric vs Non-Parametric Methods

Datasets can be represented as a set of points in a high-dimensional space; a data point with n features $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ can be represented with the feature vector $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ in n-dimensional space. Parametric methods of supervised learning attempt to model the data using these features, while non-parametric (also known as instance-based) methods do not.

4.1.1 Approximation

Parametric methods use parameters to create global approximations. Non-parametric methods instead create approximations based on local data.

4.1.2 Efficiency

Parametric methods do most of their computation beforehand, and the summarize their results in a set of parameters. Non-parametric methods tend to have a shorter training time but a longer query answering time.

4.2 K-Nearest Neighbors

K-nearest neighbors (KNN) is a common non-parametric method. The idea is to predict the value of a new point based on the values of the K most similar (i.e. closest) points.

4.2.1 Implementation

A common implementation of KNN involves looping through all N points in a training set and computing their distance to some point x. Then the K nearest points are selected. This process can be sped up by storing the data points in a data structure that helps facilitate distance-based search (e.g. a k-d tree).

4.2.2 Distance Function

"Nearby" means of minimal distance, which is commonly defined by Euclidean distance. Other distance functions $d(x, x')$ can be used, though must meet the following conditions:

- $d(x, x') = d(x', x)$ (i.e. symmetric)
- $d(x, x) = 0$ (i.e. definite)
- $d(a, c) \leq d(a, b) + d(b, c)$ (i.e. triangle inequality holds)

4.2.3 Decision Boundaries

Decision boundaries define the borders of a single classification of input. These boundaries are formed of sections of straight lines that are equidistant to two points of different classes. A highly jagged line is an indicator of overfitting, while a simple line is an indicator of underfitting.

4.2.4 Selection of K

The selection of the value of K is a bias-variance tradeoff. Low values of K have high variance but low bias, while high values of K have low variance but high bias. High-values of K result in smoother decision boundaries, which can be a sign of underfitting, and vice versa.

K can be selected experimentally by evaluating the performance for different values of K through cross-validation or against a testing set. In theory, as the number of training examples approaches infinity, the error rate of a 1NN classifier is at worst twice that of the Bayes Optimal Classifier.

4.2.5 Pre-Processing

Some common forms of pre-processing for KNN include:

- Removing undesirable inputs. Common removal methods are:
  - Editing methods, which involve eliminating noisy points of data.
  - Condensation methods, which involve selecting a subset of data that produces the same or very similar classifications.
- Use custom weights for each feature (not all features may be equally relevant for the situation)

4.2.6 Distance-Weighted Nearest Neighbor

A common problem with KNN is that it can be sensitive to small changes in the training data. One way to mitigate with drawback is to compute a weight for each neighbor based on its distance (e.g. through a Gaussian distribution), and this weight determines how much of an influence that point's value has. This differs from standard KNN which weighs the values of the K nearest neighbors equally and ignores all other values.

4.2.7 High Dimensionality

In uniformly distributed high-dimensional spaces, distances between points tend to be roughly equal, since there are so many features that changing a few features results in only a small change in distance. However, KNN can still be applied in practice for high-dimensional spaces, since data in high-dimensional spaces tends to be concentrated around certain hubs rather than uniformly distributed.

5 Statistical Learning

Data is often incomplete, indirect, or noisy. Statistical learning lets us consider forms of uncertainty to help us build better models. If we have access to the underlying probability distribution of the data, then we can form an optimal regression or classifier. In practice we typically do not know the underlying probability distributions, so we have to estimate them from the available training data. It is generally best to choose a family of parametric distributions (e.g. Gaussian or Binomial) and then determine which parameters describe the available training data the best. This is known as a density estimate and we assume that each point of training data is independently selected from the same distribution.

5.1 Bayesian Learning

Bayes' theorem describes the probability of an event H given evidence e.

$$P(H|e) = \frac{P(e|H)P(H)}{P(e)} \tag{1}$$

$$= kP(e|H)P(H) \tag{2}$$

where:

- $P(H|e)$: Posterior probability
- $P(e|P)$: Likelihood
- $P(H)$: Prior probability
- $P(e)/k$: Normalizing constant

Bayesian Learning consists of determining the posterior probability using Bayes' theorem.

Suppose we want to make a prediction about an unknown quantity X we can consider the hypothesis space which represents all possible models h_i to predict the scenario.

$$P(X|\boldsymbol{e}) = \sum_i P(X|e, h_i)P(h_i|e) \tag{3}$$

$$= \sum_i P(X|h_i)P(h_i|e) \tag{4}$$

This prediction yields the weighted combination of all the hypothesis' in the hypothesis space based on it's likelihood from the evidence. The prior $P(h_i|e)$ is yields the weight for each hypothesis and $P(X|h_i)$ yields the likelihood of the hypothesis for the unknown quantity X.

Bayesian probability is:

- Optimal: give a prior probability, no prediction is correct more often than the Bayesian prediction.
- Overfitting-free: all hypothesis are weighted and considered, eliminating overfitting.

One of the constraints of bayesian learning is that it can be intractable when the hypothesis space grows very large, often as a result of approximating a continuous hypothesis space with many discrete hypothesis. This requires us to approximate Bayesian Learning.

5.2 Approximate Bayesian Learning

5.2.1 Maximum a Posteriori

Maximum a Posteriori (MAP) makes predictions based on only the most probable hypothesis $h_{MAP} = argmax_{h_i} P(h_i|e)$. This differs from Bayesian learning, which makes predictions for all hypothesis weighted by their probability. MAP and Bayesian learning predictions tend to converge as the amount of data increases, and overfitting can be mitigated by giving complex hypothesis a low prior probability. However, finding h_{MAP} may be difficult or intractable.

5.2.2 Maximum Likelihood

Maximum Likelihood (ML) simplifies MAP by assuming uniform prior probabilities and then makes a prediction based on the most probable hypothesis h_{ML}. ML tends to be less

accurate than MAP and Bayesian predictions, it is also subject to overfitting due to the prior probabilities being uniform. Finding h_{ML} is easier than finding h_{MAP} since finding h_{ML} for $P(e|h)$ is equivalent to calculating it for $argmax_h \sum_n logP(e_n|h)$.

5.3 Bayesian Linear Regression

Instead of taking the hypothesis $\boldsymbol{w}$ that maximizes the posterior we can compute the posterior and work with that directly as follows:

$$\begin{aligned} P(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X}) &= \frac{P(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X})P(\boldsymbol{w}|\boldsymbol{X})}{P(\boldsymbol{y}|\boldsymbol{x})} \\ &= ke^{-\frac{1}{2}(\boldsymbol{w}-\overline{\boldsymbol{w}})^T \boldsymbol{A}(\boldsymbol{w}-\overline{\boldsymbol{w}})} \\ &= N(\overline{\boldsymbol{w}}, \boldsymbol{A^{-1}}) \end{aligned}$$

where

$$\begin{aligned} \overline{w} &= \sigma^{-2}\boldsymbol{A^{-1}}\overline{\boldsymbol{X}}\boldsymbol{y} \\ A &= \sigma^{-2}\overline{\boldsymbol{X}}\overline{\boldsymbol{X}}^T + \boldsymbol{\Sigma^{-1}} \end{aligned}$$

5.3.1 Prediction

Let us consider an input $\boldsymbol{x_*}$ for which we want a corresponding prediction y_*.

$$\begin{aligned} P(y_*|\overline{\boldsymbol{x_*}}, \overline{\boldsymbol{X}}, \boldsymbol{y}) &= \int_{\boldsymbol{w}} P(y_*|\overline{\boldsymbol{x_*}}, \boldsymbol{w})P(\boldsymbol{w}|\overline{\boldsymbol{X}}, \boldsymbol{y})d\boldsymbol{w} \\ &= k\int_{\boldsymbol{w}} e^{-\frac{(y_*-\overline{\boldsymbol{x}}^T\boldsymbol{w})^2}{2\sigma^2}} ke^{-\frac{1}{2}(\boldsymbol{w}-\overline{\boldsymbol{w}})^T\boldsymbol{A}(\boldsymbol{w}-\overline{\boldsymbol{w}})}d\boldsymbol{w} \\ &= N(\overline{\boldsymbol{x_*}}^T\boldsymbol{A}^{-1}\overline{\boldsymbol{X}}\boldsymbol{y}, \overline{\boldsymbol{x_*}}^T\boldsymbol{A}^{-1}\overline{\boldsymbol{x_*}}) \end{aligned}$$

This gives us a gaussian distribution of the solution. Generally for the prediction we take the mean of the distribution.

5.4 Noisy Linear Regression

Linear Regression data is often noisy and isn't distributed in a perfectly straight line.

$$y = f(\overline{x}) + \varepsilon$$

Now assuming our noise ε is a Gaussian distribution (good in practice and mathematically) then we get the likelihood distribution:

$$\begin{aligned} P(\boldsymbol{y}|X, \boldsymbol{w}, \sigma) &= N(\boldsymbol{y}|\boldsymbol{w}^T X, \sigma^2) \\ &= \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_n-\boldsymbol{w}^T\boldsymbol{x}_n)^2}{2\sigma^2}} \end{aligned}$$

5.4.1 Maximum Likelihood Solution

We can apply maximum likelihood to this and find the best $\boldsymbol{w}^*$ by maximizing the likelihood of the data.

$$\begin{aligned}\boldsymbol{w}^* &= argmax_w P(\boldsymbol{y}|X, \boldsymbol{w}, \sigma) \\ &= argmax_w \prod_n e^{-\frac{(y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2}{2\sigma^2}} \\ &= argmax_w \sum_n -\frac{(y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2}{2\sigma^2} \\ &= argmin_w \sum_n (y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2\end{aligned}$$

This leads us to least square problem derived in the Linear Regression section using the Mean Squared Error.

5.4.2 Maximum A Posteriori Solution

Alternatively we can apply MAP to our noisy linear regression problem and find $\boldsymbol{w}^*$ with the highest posterior probability (most probable hypothesis).

Gaussian Prior:

$$P(\boldsymbol{w}) = N(0, \boldsymbol{\Sigma})$$

Posterior:

$$\begin{aligned}P(\boldsymbol{w}|X, \boldsymbol{y}) &\propto P(\boldsymbol{w}) P(\boldsymbol{y}|X, \boldsymbol{w}) \\ &= k e^{-\frac{\boldsymbol{w}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{w}}{2}} e^{-\frac{\sum_n (y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2}{2\sigma^2}}\end{aligned}$$

We can now simplify this to an optimization problem of finding

$$\begin{aligned}\boldsymbol{w}^* &= argmax_{\boldsymbol{w}} P(\boldsymbol{w}|X, \boldsymbol{y}) \\ &= argmax_{\boldsymbol{w}} -\sum_n (y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2 - \boldsymbol{w}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{w} \\ &= argmin_{\boldsymbol{w}} \sum_n (y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2 + \boldsymbol{w}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{w}\end{aligned}$$

Let $\boldsymbol{\Sigma}^{-1} = \lambda I$ then

$$\boldsymbol{w}^* = argmin_{\boldsymbol{w}} \sum_n (y_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2 + \lambda \|\boldsymbol{w}\|^2$$

This is the ridge regularized least square problem that reduces overfitting.

5.5 Mixture of Gaussians

Now we consider the probabilistic generative model for classification. We can compute the posterior $P(C|\boldsymbol{x})$ according to Bayes' theorem to estimate the probability of the class for a given data point. Here we are using Bayes theorem for inference rather than for Bayesian learning (estimating parameters of a model).

$$\begin{aligned} P(C|\boldsymbol{x}) &= \frac{P(\boldsymbol{x}|C)P(C)}{\sum_C P(\boldsymbol{x}|C)P(C)} \\ &= kP(\boldsymbol{x}|C)P(C) \end{aligned}$$

where:

- $P(C)$: Prior probability of class C
- $P(\boldsymbol{x}|C)$: class conditional distribution of $\boldsymbol{x}$

with the following assumptions:

- In classification the number of classes is finite, so a natural prior $P(C)$ is the multinomial $P(C = c_k) = \pi_k$
- when $\boldsymbol{x} \in \mathbb{R}$ then it is often ok to assume that $P(\boldsymbol{x}|C)$ is Gaussian.
- Assume the same covariance matrix $\boldsymbol{\Sigma}$ is used for each class.

From our assumptions we get

$$P(\boldsymbol{x}|c_k) \propto e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)}$$

5.5.1 Binary Classification

Subbing our assumptions into Bayes theorem for binary classification and simplifying, we get the following posterior distribution for classes c_k, c_j.

$$P(c_k|\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}+w_0}}$$

where:

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_j) \\ w_0 &= \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \frac{1}{2}\boldsymbol{\mu}_j^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j + ln\frac{\pi_k}{\pi_j} \end{aligned}$$

We can observe that this equation is the logistic sigmoid and we can draw the class boundary at $\sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0) = 0.5$.

5.5.2 Multinomial Classification

Now similarly for a multi-class problem with K classes we get.

$$P(c_k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_k^T \boldsymbol{x}}}{\sum_j e^{\boldsymbol{w}_j^T \boldsymbol{x}}}$$

where

$$\boldsymbol{w}_k^T = (-\frac{1}{2}\boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + ln\pi_k, \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1})$$

This process can be extrapolated for classes that aren't all distributed with a gaussian distribution (e.g exponential, poisson, bernoulli etc ...). We can see that this is a specific case of the softmax distribution which is a generalization of the sigmoid and is discussed in further detail in the next section.

5.5.3 Parameter Estimation

Let $\pi = P(y = C_1)$ and $1 - \pi = P(y = C_2)$ where $P(x|C_1) = N(x|\mu_1, \Sigma)$ and $P(x|C_2) = N(x|\mu_2, \Sigma)$. In order to actually use bayesian inference to get the classification probability of our input data, we need to learn the parameters π, μ_1, μ_2 and Σ. We can estimate the parameters by maximum likelihood, maximum a posteriori or bayesian learning. This example will demonstrate using maximum likelihood to learn these parameters.

We can express the Likelihood of our training set as $L(\boldsymbol{X}, \boldsymbol{y}) = P(\boldsymbol{X}, \boldsymbol{y}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma})$. We want to maximize the likelihood in order to use Bayes inference.

$$L(\boldsymbol{X}, \boldsymbol{y}) = \prod_n [\pi|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}]^{y_n}[(1-\pi)|N(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})]^{1-y_n}$$

Taking the log we can turn this into an optimization problem of finding

$$argmax_{\pi,\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\boldsymbol{\Sigma}} \sum_n y_n[ln\pi - \frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_1)] \\ + (1 - y_n)[ln\pi - \frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_2)]$$

Now to estimate π (probability of class) we can take:

$$0 = \frac{\partial lnL(X, y)}{\partial \pi}$$
$$\pi = \frac{\sum_n y_n}{N}$$

Now to estimate μ_1 and μ_2 (mean of classes) we can take:

$$0 = \frac{\partial lnL(X, y)}{\partial \mu_1}$$
$$\mu_1 = \frac{\sum_n y_n x_n}{N_1}$$

and

$$0 = \frac{\partial lnL(X, y)}{\partial \mu_2}$$
$$\mu_2 = \frac{\sum_n (1 - y_n) x_n}{N_2}$$

Now to estimate Σ (covariance matrix) we can take:

$$0 = \frac{\partial lnL(X, y)}{\partial \Sigma}$$
$$\Sigma = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2$$

where S_k are the empirival covariance matrices of the class k

$$S_1 = \frac{1}{N_1} \sum_{n \in C_1} (x_n - \mu_1)(x_n - \mu_1)^T$$
$$S_2 = \frac{1}{N_1} \sum_{n \in C_2} (x_n - \mu_2)(x_n - \mu_2)^T$$

6 Linear Models

6.1 Linear Regression

6.1.1 Formulation

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. Our main objective is to generate a line that minimizes the distance from the line to all of data points. This is essentially minimizing the error and maximizing our prediction accuracy.

6.1.2 Simple Regression

A simple two variable linear regression uses the slope-intercept form, where m and b are the variables our algorithm will try to "learn". x represents our input data and y represents the prediction.

$$y = mx + b$$

6.1.3 Multivariable Regression

Often times there are more than one feature in the data and we need a more complex multi-variable linear equation as our hypothesis. We can represent our hypothesis with the follow multi-variable linear equation, where $\boldsymbol{w}$ are the weights and $\boldsymbol{x}$ is the input data.

$$\begin{aligned} h_{\boldsymbol{w}}(\boldsymbol{x}) &= w_0 x_0 + w_1 x_1 w_2 x_2 + ... + w_n x_n \\ &= \boldsymbol{w}^T \boldsymbol{x} \end{aligned}$$

6.1.4 Cost Function

To predict based on a dataset we first need to learn the weights that minimize the mean squared error (euclidean loss) of our hypothesis. We can define the following to be our cost function to minimize with m being the number of data points and i being the i^{th} training example. This can also be proven by applying maximum likelihood with gaussian noise. Similarly we can come up with the regularized least square problem by applying maximum a posteriori on noisy linear regression.

$$\begin{aligned} J(\boldsymbol{w}) &= \frac{1}{2m} \sum_{i=1}^{m} (h_{\boldsymbol{w}}(\boldsymbol{x}^i) - y^i)^2 \\ &= \frac{1}{2m} (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y}) \end{aligned}$$

6.1.5 Gradient Descent Solution

Now to solve for $\boldsymbol{w}$ we can use Gradient Descent and iteratively update $\boldsymbol{w}$ until it converges. We get the slope of the cost function to be:

$$\frac{\partial J\boldsymbol{w}}{\partial \boldsymbol{w}_j} = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{w}^T \boldsymbol{x}^i - y^i) \boldsymbol{x}_j^i$$

now applying a step α we can iteratively change $\boldsymbol{w}$ until it reaches the global minima.

$$\boldsymbol{w}_j := \boldsymbol{w}_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{w}}(\boldsymbol{x}^i) - y^i)$$

6.1.6 Normal Equation Solution

The closed form solution to the linear system in $\boldsymbol{w}$

$$\frac{\partial J\boldsymbol{w}}{\partial \boldsymbol{w}_j} = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{w}^T \boldsymbol{x}^i - y^i) \boldsymbol{x}_j^i = 0$$

writing this as a linear system in w we get $A\boldsymbol{w} = b$ where

$$A = \sum_{n=1}^{N} (\boldsymbol{x}_n \boldsymbol{x}_n^T) \text{ and } b = \sum_{n=1}^{N} (\boldsymbol{x_n} y_n)$$

so we can solve for $\boldsymbol{w} = \boldsymbol{A}^{-1}\boldsymbol{b}$ and get the following vectorized solution.

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y}$$

6.2 Logistic Regression

6.2.1 Formulation

Logistic regression is an algorithm used for classification. It is used to estimate the probability that an instance belongs to a particular class. If the estimated probability is greater than 50%, then the model predicts the instance belongs to that class, and otherwise it predicts it does not. Logistic Regression is form of discriminative learning as it attempts to model $P(c_k|\boldsymbol{x})$ directly, this is unlike generative learning where $P(c_k)$ and $P(\boldsymbol{x}|c_k)$ are found by max likelihood and $P(c_k|\boldsymbol{x})$ by Bayesian Inference.

6.2.2 Prediction

Logistic Regression computers the weighted sum of the input features (plus a bias term) and outputs the logistic (Sigmoid Function) of the result. The hypothesis is given by

$$p = h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

Once the Logistic Regression model has estimated the probability that an instance *boldsymbolx* belongs to the positive class, it can make its prediction y easily.

$$y = \begin{cases} 0 & p < 0.5 \\ 1 & p \geq 0.5 \end{cases}$$

6.2.3 Cost Function

The objective of training the model is such that the model estimates high probabilities for positive instances ($y = 1$) and low probabilities for negative instances ($y = 0$). This concept is captured through the cost function shown below.

$$J(\boldsymbol{w}) = \begin{cases} -log(p) & y = 1 \\ -log(1-p) & y = 0 \end{cases}$$

This makes intuitive sense because $-log(t)$ grows very large when t approaches 0, so the cost will be large if the model estimates a probability close to 0 for a positive instance. The cost will also be very large if the model estimates a probability close to 1 for a negative instance. On the other hand $-log(t)$ is close to 0 when t is close to 1, so the cost will be close to 0 if the estimated probability is close to 0 for a negative instance or close to 1 for a positive instance.

We can express the cost as a single expression called the *log loss*.

$$J(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})) + (1 - y^{(i)}log(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))]$$

6.2.4 Solution

Unfortunately, there is no known closed-form solution to compute the value of $\boldsymbol{w}$ that minimizes the cost function. The cost function however, is convex, so Gradient Descent or any other optimization algorithm is guaranteed to find the global minimum. The gradient can be expressed as:

$$\frac{\partial J\boldsymbol{w}}{\partial \boldsymbol{w}_j} = \frac{1}{m}\sum_{i=1}^{m}(\sigma(\boldsymbol{w}^T\boldsymbol{x}) - y^{(i)}x_j^{(i)})$$

Some faster more sophisticated methods are

- Conjugate Gradient
- BFGS
- L-BFGS

6.2.5 Softmax Regression

The Logistic Regression model can be generalized to support multiple classes. When given an instance $\boldsymbol{x}$, the Softmax Regression model computes a score $f_k(\boldsymbol{x})$ for each class k, then estimates the probability of each class by applying the *softmax function* to the scores.

$$f_k(\boldsymbol{x}) = \boldsymbol{w}_k^T\boldsymbol{x}$$

Once the score of every class for the instance $\boldsymbol{x}$ is computed, you can estimate the probability π_k that the instance belongs to class k. The function computes the exponential of each score, the normalizes them.

$$\pi_k = P(y^{(i)} = k|\boldsymbol{x}^{(i)}, \boldsymbol{w}) = \frac{e^{f_k(\boldsymbol{x})}}{\sum_{j=1}^{K} e^{f_j(\boldsymbol{x})}}$$

The Softmax Regression classifier predicts the class with the highest estimated probability.

The cost function associated with the Softmax Regression Classifier is the Cross Entropy cost function; it penalizes the model when it estimates a low probability for a target class. Cross entropy is used to measure how well a set of estimates class probabilities matches the target class. The cost function is represented as such

$$J(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(\pi_k^{(i)})$$

with gradient vector

$$\nabla_{w_k} J(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} (\pi_k^{(i)} - y_k^{(i)}) \boldsymbol{x}^{(i)}$$

that can be paired with an optimization algorithm to solve.

6.3 Generalized Linear Models

Often times our data won't be linear and it could be of a higher degree polynomial or a completely different distribution altogether. We can turn this non-linear problem into a linear regression problem by mapping the data to a different vector space using a basis function.

To demonstrate, let us consider Linear Regression on a nonlinear N x 1 (feature) dataset. Let ϕ denote the polynomial basis function where $\phi_j(\boldsymbol{x}) = x^j$. Then we can express our hypothesis as:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0\phi(x) + w_1\phi_1(x) + w_2\phi_2(x) + ... + w_3\phi_3(x)$$

A dataset with 3 features with a polynomial basis would have a hypothesis as such

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1^2x_2 + w_6x_1x_2^2 + w_7x_1^2x_2^2 + w_8x_1^3 + w_9x_2^3$$

This can then be extrapolated to logistic regression and n-features. Some commonly used basis functions are:

- Polynomial: $\phi_j(\boldsymbol{x}) = x^j$
- Gaussian: $\phi_j(\boldsymbol{x}) = e^{(\frac{x-\mu_j}{2s^2})}$
- Sigmoid: $\phi_j(\boldsymbol{x}) = \sigma(\frac{x-\mu_j}{s})$
- Fourier Basis, Wavelets, etc ...

6.4 Regularization

Small outliers can drastically change our values of $\boldsymbol{w}$ so rely on regularization to reduce overfitting. Polynomial models can be easily regularized by reducing the number of polynomial degrees. For a linear model, regularization is typically achieved by constraining the weights of the model. The regularization term should only be added to the cost function during training. Once the model is trained, the non-regularized cost should be used to measure the model's performance. The bias term w_0 is not regularized.

6.4.1 Ridge Regression

Ridge Regression (Tikhonov Regularization) is a regularized version of Linear regression with a regularization term of $\frac{\lambda}{2}\|w\|_2^2$ (l_2-norm) added to the cost function. This forces the

learning algorithm to fit the data but also keep the model weights as small as possible. The hyperparameter λ controls how much you want to regularize the model.

$$J(\boldsymbol{w}) = ERROR(\boldsymbol{w}) + \frac{\lambda}{2}\|w\|_2^2$$

6.4.2 Lasso Regression

Least Absolute Shrinkage and Selection Operator Regression is another regularized version of Linear Regression, it adds a regularization term to the cost function but uses the l_1 norm of the weight vector instead of half the square of the l_2 norm.

$$J(\boldsymbol{w}) = ERROR(\boldsymbol{w}) + \lambda \sum_{i=1}^{n} |w_i|$$

An important characteristic of Lasso Regression is that it tends to eliminate the weights of the least important features (i.e, set them to zero). Lasso Regression automatically performs feature selection and outputs a *sparse model*.

6.4.3 Elastic Net

Elastic Net is a middle ground between Ridge Regression and Lasso Regression. The regularization term is a simple mix of both Ridge and Lasso's regularization terms and you can control the mix ratio r. When $r = 0$, Elastic Net is equivalent to Ridge Regression, and when $r = 1$, it is equivalent to Lasso Regression.

$$J(\boldsymbol{w}) = ERROR(\boldsymbol{w}) + \frac{(1-r)\lambda}{2}\|w\|_2^2 + r\lambda \sum_{i=1}^{n} |w_i|$$

6.4.4 Early Stopping

Early Stopping is a different way to regularize iterative learning algorithms such as Gradient Descent. This method aims to stop training as soon as the validation error reaches a minimum. For all convex optimization problems there will be a global minima, once that global minima is reached the curve will start going up. This proposes to stop as soong as we reach the minimum.