

1 Supervised Learning

1.1 Part I: Linear Regression

Let θ_i 's be the **parameters** (also called **weights**) parameterizing the space of linear functions mapping from \mathbb{X} to \mathbb{Y} . We make the notation, with $x_0 = 1$,

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x \quad (1)$$

as a hypothesis h where on the right-hand side above we are viewing θ and x both as vectors, and here n is the number of input variables (not counting x_0).

We now define the **cost function**:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2. \quad (2)$$

1.1.1 LMS algorithm

gradient descent We consider the **gradient descent** algorithm, which starts with some initial θ , and repeatedly performs update:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta). \quad (3)$$

Here, α is called the **learning rate**. In order to implement this algorithm, we need to work out what is the partial derivative term on the right hand side. Let's first work it out for the case of if we have only one training example (x, y) , so that we can neglect the sum in the definition of J . We have:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\ &= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_\theta(x) - y) x_j \end{aligned}$$

Thus, for a single training example, this gives the update rule:

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}. \quad (4)$$

batch gradient descent This algorithm looks at every example in the entire training set on every step.

repeat

$$\theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

until it is convergence

stochastic gradient descent This algorithm updates the parameters according to the gradient of the error with respect to that single training example only.

```

loop
  for  $i = 1$  to  $m$  do
     $\theta_j \leftarrow \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$ 
  end for
end loop

```

1.2 The normal equations

In this section, we will discuss a second way of minimizing J by using derivatives.

1.2.1 Matrix derivatives

$\nabla_A f(A)$ For a function $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$, we define the derivative of f with respect to A to be:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} \quad (5)$$

trace operation We define the **trace** operator as:

$$\text{tr} A = \sum_{i=1}^n A_{ii} \quad (6)$$

We'll now prove that $\text{tr}(AB) = \text{tr}(BA)$ if AB is square:

$$\begin{aligned} \text{tr}(AB) &= \sum_{i=1}^n (AB)_{ii} \\ &= \sum_{i=1}^n \sum_{j=1}^n a_{ij} b_{ji} \\ &= \sum_{j=1}^n \sum_{i=1}^n b_{ji} a_{ij} \\ &= \sum_{j=1}^n (BA)_{jj} \\ &= \text{tr}(BA) \end{aligned}$$

Followings are some more facts of matrix derivatives for if both A and B are square matrices:

$$\nabla_A \text{tr} AB = B^T \quad (7)$$

$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T \quad (8)$$

$$\nabla_A \text{tr} ABA^T C = CAB + C^T AB^T \quad (9)$$

$$\nabla_A |A| = |A| (A^{-1})^T \quad (10)$$

1.2.2 Least squares revisited

design matrix Given a training set, define the **design matrix** X to be the m -by- $(n+1)$ matrix (including the intercept term) that contains the training examples' input values in its row:

$$X = \begin{bmatrix} - (x^{(1)})^T & - \\ - (x^{(2)})^T & - \\ \vdots & \\ - (x^{(m)})^T & - \end{bmatrix} \quad (11)$$

Let \vec{y} be the m -dimensional vector containing all the target values from the training set:

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad (12)$$

Now, since $h_{\theta}(x^{(i)})^T \theta$, we can easily verify that

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T \theta \\ (x^{(2)})^T \theta \\ \vdots \\ (x^{(m)})^T \theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_{\theta}(x^{(1)}) - y^{(1)} \\ h_{\theta}(x^{(2)}) - y^{(2)} \\ \vdots \\ h_{\theta}(x^{(m)}) - y^{(m)} \end{bmatrix}. \end{aligned}$$

Thus, using the fact that for a vector z , we have that $z^T z = \sum_i z_i^2$:

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

Finally, to minimize J , we find its derivative with respect to θ :

$$\begin{aligned}
\nabla_{\theta} J(\theta) &= \nabla_{\theta} \frac{1}{2} (X\theta - \vec{y})^T (X\theta - \vec{y}) \\
&= \frac{1}{2} \nabla_{\theta} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y}) \\
&= \frac{1}{2} \nabla_{\theta} \text{tr} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y}) \\
&= \frac{1}{2} \nabla_{\theta} (\text{tr} \theta^T X^T X \theta - 2 \text{tr} \vec{y}) \\
&= \frac{1}{2} (X^T X \theta + X^T X \theta - 2 X^T \vec{y}) \\
&= X^T X \theta - X^T \vec{y}
\end{aligned}$$

To minimize J , we set its derivative to zero, and obtain the **normal equations**:

$$X^T X \theta = X^T \vec{y} \quad (13)$$

Thus, the value of θ that minimizes $J(\theta)$ is given in closed form by the equation

$$\theta = (X^T X)^{-1} X^T \vec{y}. \quad (14)$$

2 Statistical Learning

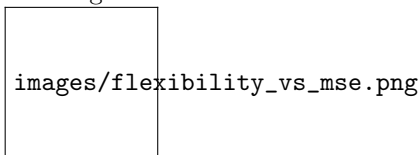
2.1 Measuring the Quantity fit

Mean squared error The book defines the *mean squared error* (MSE), given by

$$MSE = \frac{1}{n} \sum_{i=1}^n \left(y_i - \hat{f}(x_i) \right)^2$$

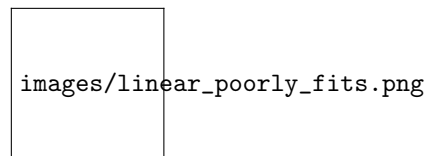
where $\hat{f}(x_i)$ is the prediction that \hat{f} gives for the i th observation. This value will be small if the predicted responses are very close to the true responses. Otherwise, the predicted value will be substantially high due to possibly bad predictors.

degree of freedom In short, **degree of freedom** describes the flexibility of a curve. The higher value of degree of freedom results in a highly fluctuating curve.



This graph shows an example that the true f is approximately linear. We observe that the training MSE decreases monotonically as the model flexibility increases, and that there is a U-shape in the test MSE. However, because the truth is so close to linear, the test MSE only decreases slightly before increasing again, so that **the orange least squares fit is substantially better than the highly flexible green curve**.

2.2 The Bias-Variance Trade-Off



This graphs shows an example that the linear least square fit (as shown in the orange line) poorly fits the true f .

Expected test MSE It is possible to show that the **expected test MSE**, for a given value x_0 , can always be decomposed into the sum of three fundamental quantities: the *variance* of $\hat{f}(x_0)$, the squared *bias* of $\hat{f}(x_0)$, and the variance of the error terms ϵ . That is,

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + \left[\text{Bias}(\hat{f}(x_0))\right]^2 + \text{Var}(\epsilon).$$

We need to select a statistical learning method that simultaneously achieves low variance and low bias.

Variance Variance in this context refers to the amount by which \hat{f} would change if we estimated it using a different training set. Since the training data are used to fir the statistical learning method, different training data sets will result in a different \hat{f} . But ideally the estimate for f should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in \hat{f} . In general, more flexible statistical methods have higher variance.

bias On the other hand, *bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. For example, linear regression assumes that there is a linear relationship between Y and X_1, X_2, \dots, X_p . It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubted result in some bias in the estimate of f . **As a general rule, as we use more flexible methods, the variance will increase and the bias will decrease.**

2.3 The Classification Setting

the training error rate We define it as

$$\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i).$$

Here \hat{y}_i is the predicted class label for the i th observation using \hat{f} . And $I(y_i \neq \hat{y}_i)$ is an indicator variable that equals 1 if $y_i \neq \hat{y}_i$ and 0 otherwise. A good classifier is one for which the test error is smallest.

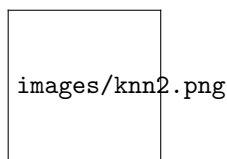
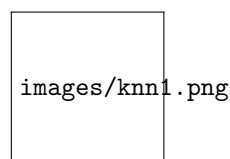
The Bayes Classifier We classify data by assigning them to their most likely class, given their predictor values. In other words, we should simply assign a test observation with predictor vector x_0 to the class j for which

$$\Pr(Y = j|X = x_0)$$

is largest. To illustrate, in a two-class problem, the Bayes classifier corresponds to predicting class one if $\Pr(Y = 1|X = x_0) > 0.5$, and class two otherwise.

K-Nearest Neighbors In theory we would always like to predict qualitative responses using the Bayes classifier. But for real data, we do not know the conditional distribution of Y given X , and so computing the Bayes classifier is impossible. K-nearest neighbors (KNN) classifier can solve such a problem. Given a positive integer K and a test observation x_0 , the KNN classifier first identifies the K points in the training data that are closest to x_0 , represented by N_0 . It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j :

$$\Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j).$$



These images show the examples of KNN classifier.

3 Classification

Ref. ISLR Book Examples:

1. A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?
2. An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.
3. On the basis of DNA sequence data or a number of patients with and without a given disease, a biologist would like to figure out which DNA mutations are deleterious (disease-causing) and which are not.

3.1 Why Not Linear Regression?

Suppose that we are trying to predict the medical condition of a patient in the emergency room on the basis of her symptoms. In this simplified example, there are three possible diagnoses. Least

square could be used to fit a linear regression model to predict Y on the basis of a set of predictors X_1, \dots, X_p .

By doing so, the problem occurs because the order of three possible outcomes matter, e.g. the question arises if the first outcome should be between other two. This implies a different relationship among the three conditions. Each of these different orders can produce fundamentally different linear models that would ultimately lead to different sets of predictions on test observations.

For a binary qualitative response, the situation is better. However, for linear regression, some of our estimates might be outside the $[0, 1]$ interval, making them hard to interpret as probabilities. Nevertheless, the predictions provide an ordering and can be interpreted as crude probability estimates.

3.2 Logistic Regression

The problem from using linear regression gives rise to the logistic model. We introduce a model:

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

which the output of the function is in the interval $[0, 1]$ for all values of X .

To fit the model above, we use a method called *maximum likelihood*, which we will discuss in the next subsection. The logistic function will always produce an *S-shaped* curve of this form, and so regardless of the value of X , we will obtain a sensible prediction.

After a bit of manipulation of the equation above, we find that

$$\begin{aligned} \frac{p(X)}{1 - p(X)} &= e^{\beta_0 + \beta_1 X} \\ \log \left(\frac{p(X)}{1 - p(X)} \right) &= \beta_0 + \beta_1 X \end{aligned}$$

which shows that the logistic regression model has a logit that is linear in X .

3.3 Estimating the Regression Coefficients

Even though we can use the least squares approach to estimate the unknown linear regression coefficients above, the more general method of *maximum likelihood* is preferred, since it has better statistical properties.

The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ of default for each individual, using the logistic model, corresponds as closely as possible to the individual's observed default status. In other words, we try to find $\hat{\beta}_0$ and $\hat{\beta}_1$ such that plugging these estimates into the model for $p(X)$ yield s a number close to one for all individuals who defaulted, and a number close to zero for all individual who did not. This intuition can be formalized using a mathematical equation called a *likelihood function*:

$$l(\beta_0, \beta_1) = \prod_{i: y_i=1} p(x_i) \cdot \prod_{i': y_{i'}=0} (1 - p(x_{i'}))$$

The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ will be chosed to maximuze this likelihood function.

3.4 Linear Discriminant Analysis

Logistic regression involves directly modeling $\Pr(Y = k|X = x)$ using the logistic function. We now consider an alternative approach to estimating these probabilities, which we model the distribution of the predictors X separately in each of the response classes Y , and then use Bayes' theorem to flip these around into estimates for $\Pr(Y = k|X = x)$. Here are several reasons of doing so:

1. When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.
2. If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
3. Linear discriminant analysis is popular when we have more than two response classes.

3.4.1 Using Bayes' Theorem for Classification

The Bayes' Theorem states that

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{i=1}^K \pi_i f_i(x)}$$

3.4.2 Linear Discriminant Analysis for $p = 1$

For now, we assume that $p = 1$, that is, we have only one predictor. We would like to obtain an estimate for $f_k(x)$ in order to estimate $p_k(x)$. We will then classify an observation to the class for which $p_k(x)$ is greatest.

In order to estimate $f_k(x)$, we make some assumptions about its form. We assume that $f_k(x)$ is normal or Gaussian. In the one-dimensional setting, the normal density takes the form

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right)$$

where μ_k and σ_k^2 are the mean and variance parameters for the k th class. For now, let further assume that $\sigma_1^2 = \dots = \sigma_K^2$: that is, there is a shared variance term across all K classes, which for simplicity we can denote by σ^2 . We then find that

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_l)^2\right)}$$

Note that π_k denotes the prior probability that an observation belongs to the k th class.

To be added.

3.4.3 Linear Discriminant Analysis for $p \geq 1$

3.4.4 Quadratic Discriminant Analysis

4 Implementation - Python

4.1 Classification

4.1.1 Machine Learning Basics

The steps in developing a machine learning application:

1. Collect data.
2. Prepare the input data.
3. Analyze the input data.
4. Make sure the data don't have garbage coming in.
5. Train the algorithm.
6. Test the algorithm.
7. Use it.

We decide to use Python because of its clear syntax and extremely easy text manipulation. A large number of people around the world are using Python, so there are abundant development and documentation about machine learning.

We will also focus on using NumPy library for doing some linear algebra. For NumPy, there are two different data types for dealing with rows and columns of numbers. Be careful of this because they look similar, but simple mathematical operations such as multiply on the two data types can have different meanings. The matrix data type behaves more like matrices in MATLAB.

4.1.2 Classification with k-Nearest Neighbors

k-Nearest Neighbors

Pros: High accuracy, insensitive to outliers, no assumptions about data

Cons: Computationally expensive, requires a lot of memory

Works with: Numeric values, nominal values

General approach to kNN

1. Collect: Any method.
2. Prepare: Numeric values are needed for a distance calculation. A structured data format is best.
3. Analyze: Any method.
4. Train: Does not apply to the kNN algorithm.
5. Test: Calculate the error rate.
6. Use: This application needs to get some input data and output structured numeric values. Next, the application runs the kNN algorithm on this input data and determines which class the input data should belong to. The application then takes some action on the calculated class.

Prepare: Importing data with Python

5 Image Classification (CS231n)

Motivation This is one of the core problems in Computer Vision that, despite its simplicity, has a very large variety of practical applications. Moreover, as we will see later in the course, many other seemingly distinct Computer Vision tasks (such as object detection, segmentation) can be reduced to image classification.

Example You want to create a classification model that takes a single image and assigns probabilities to 4 labels, cat, dog, hat, mug. An image is represented as one large 3-dimensional array of numbers (Width x Height x RGB). Each number is an integer that ranges from 0 (black) to 255 (white).

Challenges The following challenges are worth considered:


1. **Viewpoint variation:** A single instance of an object can be oriented in many ways with respect to the camera.
2. **Scale Variation:** Visual classes often exhibit variation in their size.
3. **Deformation:** Many objects of interest are not rigid bodies and can be deformed in extreme ways.
4. **Occlusion:** The objects of interest can be occluded. Sometimes only a small portion of an object could be visible.
5. **Illumination conditions:** The effects of illumination are drastic on the pixel level.
6. **Background clutter:** The objects of interest may blend into their environment, making them hard to identify.
7. **Intra-class variation:** The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

Data-driven approach It is not obvious how one can write an algorithm to classify cat, compared to an algorithm to sort numbers. Therefore, instead of trying to specify what every one of the categories of interest look like directly in code, the approach that we will take is not unlike one you would take with a child: we're going to provide the computer with many examples of each class and then develop learning algorithm that look at these examples and learn about the visual appearance of each class. This approach is referred to as a **data-driven approach**.

5.1 L1/L2 distance

Now we'll introduce **L1/L2 distance**. One of the simplest possibilities is to compare the images pixel by pixel and add up all the difference. In other words, given two images and representing them as vectors I_1, I_2 , a reasonable choice for comparing them might be the **L1 distance**:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

 images/l1distance.jpg

Another approach for computing distances between vectors is to use the **L2 distance**, which has the geometric interpretation of computing the euclidean distance between two vectors:


$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

5.2 k-Nearest Neighbor Classifier

The idea is very simple: finding the top **k** closest images, and have them vote on the label of the test image. In particular, when $k = 1$, it is the Nearest Neighbor Classifier. Intuitively, higher values of k have a smoothing effect that makes the classifier more resistant to outlier, e.g. higher variance. In practice, you do not know what value of k should you use? This leads to the next problem: **hyperparameter tuning**.

Validation sets for Hyperparameter tuning When working with many *unknown* variables, we often call these various choices as **hyperparameters**. They come up very often in the design of many Machine Learning algorithms that learn from data, and it is often not obvious what values one should choose. We can attempt to use many different values; however, the key idea is **we cannot use the test set for the purpose of tweaking hyperparameters** because the algorithm would lose its generalization and bring higher variance, overfitting the test set. The usual way to deal with the problem is to use validation to tune these hyperparameters and verify them with the test data.

Cross-validation the idea is that instead of arbitrarily picking the first 1000 datapoints to be the validation set and the rest as a training set, you can get better and less noisy estimate of how well a certain value of k works by iterating over different validation sets and averaging the performance across these. For example, in 5-fold cross validation, we should split the training into 5 equal folds, use 4 of them for training, and 1 for validation so that we can iterate over which fold is the validation fold, evaluate the performance, and finally average the performance across the different folds.

 images/cvplot.png

From this plot, we can see that about $k = 7$ works best on this particular dataset (corresponding to the peak in the plot). If we used more than 5 folds, we might expect to see a smoother (i.e. less noisy) curve.

5.3 Linear Classification

Intro to Linear classification In the last section, we learn about k-Nearest Neighbor (kNN) classifier which labels images by comparing them to (annotated) images from the training set. As we saw, kNN has a number of disadvantages:

1. The classifier must remember all of the training data, which causes a problem when the amount of data is large.
2. Classifying a test image is expensive since it requires a comparison to all training images.

Thus, we are now going to develop a more powerful approach to image classification that we'll eventually naturally extend to entire Neural Networks and Convolutional Neural Networks. This approach will have two major components: a **score function** that maps the raw data to class scores, and a **loss function** that quantifies the agreement between the predicted scores and the ground truth labels. We'll then cast this as an optimization problem in which we will minimize the loss function with respect to the parameters of the score function.

Linear score function Let's assume a training dataset of images $x_i \in \mathbb{R}^D$, each associated with a label y_i . Here $i = 1, \dots, N$ and $y_i \in 1, \dots, K$. That is, we have N examples (each with a dimensionality D) and K distinct categories. For example, in CIFAR-10 we have a training set of $N = 50000$ images, each with $D = 32 \times 32 \times 3 = 3072$ pixels, and $K = 10$, since there are 10 distinct classes. We will now define the score function $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$ that maps the raw image pixels to class scores.


Linear classifier We'll start out with arguably the simplest possible function, a linear mapping:

$$f(x_i, W, b) = Wx_i + b$$

In the above equation, we are assuming that the image x_i has all of its pixels flattened out to a single column vector of shape $[D \times 1]$. The matrix W (of size $[K \times D]$), and the vector b , usually called the **bias vector**, (of size $[K \times 1]$) are the **parameters** of the function.

It's important to note that we have control over the setting of the parameters **W**, **b**. Our goal is to set these in such a way that the computed scores match the ground truth labels across the whole training set. Intuitively, we wish that the correct class has a score that is higher than the scores of incorrect classes. In addition, note that to classify the test image involves a single matrix multiplication and addition is significantly faster than to compare a test image to all training images.

Interpreting a linear classifier Notice that a linear classifier computes the score of a class as a weighted sum of all of its pixel values across all 3 of its color channels. Depending on precisely what values we set for these weights, the function has the capacity to like or dislike certain colors at certain positions in the image. For instance, you can imagine that the "ship" class might be more likely if there is a lot of blue on the sides of an image (which could correspond to water).

 `images/imagemap.jpg`

Another interpretation for the weights W is that each row of W corresponds to a *template* (or sometimes also called a *prototype*) for one of the classes. The score of each class for an image is then obtained by comparing each template with the image using an inner product (or *dot product*) one by one to find the one that "fits" best. Another way to think of it is that we are still effectively doing Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class (although we will learn it, and it does not necessarily have to be one of the images in the training set), and we use the (negative) inner product as the distance instead of the L1 or L2 distance.

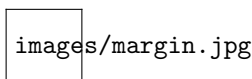
Loss function In addition to have a function for measuring the score corresponding to each class, we also want to measure our unhappiness with outcomes so that it introduces a **loss function** (or sometimes also referred to as the **cost function** or the **objective**). Intuitively, the loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

5.3.1 Multiclass SVM

One of two most commonly used loss function is the **Multiclass Support Vector Machine** (SVM) loss, given that $f(x_i, W)$ is the scores for each class:

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)]$$

Visually, we want the loss function to differentiate the correct class from other incorrect classes by parameter Δ as shown in the figure below.



Regularization There is one minor bug with the loss function presented above. Suppose that we have a dataset and a set of parameters W that correctly classify every example. The issue is that this set of W is not necessarily unique: there might be many similar W that correctly classify the examples as well. For instance, any multiple of these parameters, λW , where $\lambda > 1$. This brings up to find a way to penalize the set of parameters that are not unique by favoring the parameters whose values are smaller. By using the **regularization penalty** $R(W)$, the *loss function* will be changed to

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

where the latter term is **regularization loss**.

5.3.2 Softmax classifier

The other popular choice rather than SVM is the **Softmax classifier**, which has a different loss function that makes use of the logistic regression classifier. We replace the *hinge loss* with a **cross-entropy loss** that has the form:

$$L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \quad \text{or equivalently} \quad L_i = -f_{y_i} + \log \sum_j e^{f_j}$$

where we are using the notation f_j to mean the j -th element of the vector of class scores f . Oftentimes, $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is also called the **softmax function**: it takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one.

Practical issues: Numeric stability When you're writing code for computing Softmax function in practice, the intermediate terms $e^{f_{y_i}}$ and $\sum_j e^{f_j}$ may vary very large due to the exponentials. Thus, dividing large numbers can be numerically unstable, so it is important to use a normalization trick. Notice that this expression is mathematically equivalent:

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{C e^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i} + \log C}}{\sum_j e^{f_j + \log C}}$$

From the expression on the right, we can choose the value of C that can solve the very large value of the exponentials. The simplest way is to set $\log C = -\max_j f_j$.

5.3.3 SVM vs Softmax

Basically, SVM will give a score corresponding to each class of the possible output; however, Softmax will give the "probabilities" that can give a sense of better thought of as confidences.

 images/svmvssoftmax.png

5.3.4 Optional: Deep Learning using Linear Support Vector Machines

This paper is accessible on <http://arxiv.org/pdf/1306.0239v4.pdf>.

This paper shows that simply changing the softmax activation function to linear SVMs gives significant gains on popular deep learning datasets MNIST, CIFAR-10, and the ICML 2013.

Unlike the hinge loss of a standard SVM, the loss from the L2-SVM is differentiable and penalizes errors much heavily. From the study, they believe that the performance gain is largely due to the superior regularization effects of the SVM loss function, rather than an advantage from better parameter optimization.

Multiclass SVMs We denote the output of the k -th SVM as

$$a_k(x) = w^T x$$

and the predicted class is then

$$\arg \max_k a_k(x)$$

Deep Learning with Support Vector Machines In this paper, they use L2-SVM to train deep neural nets for classification. Lower layer weights are learned by backpropagating the gradients from the top layer linear SVM. For the softmax classifier, we define the cost function, objective, as:

$$l(w) = \min_w \frac{1}{2} w^T w + C \sum_{n=1}^N \max(1 - w^T x_n t_n, 0)$$

Let the input x replaced with the penultimate activation h ,

$$\frac{\partial l(w)}{\partial h_n} = -C t_n w (\mathbb{I}\{1 > w^T h_n t_n\})$$

where \mathbb{I} is the indicator function. Likewise, for the L2-SVM, we have

$$\frac{\partial l(w)}{\partial h_n} = -2C t_n w (\max(1 - w^T h_n t_n, 0))$$

From this point, we see that backpropagation algorithm is exactly the same as the standard softmax-based deep learning networks.

Experiment: Facial Expression Recognition This challenge was hosted by the ICML 2013 workshop on representation learning, organized by the LISA at University of Montreal. The data consist of 28,709 48x48 images of faces under 7 different types of expression. The winning solution, which succeeded 69.4% on the public validation and 71.2% on the private test. This solution consists of using a simple Convolutional Neural Network with linear one-vs-all SVM at the top. Stochastic gradient descent with momentum is used for training and several models are averaged to slightly improve the generalization capabilities.