Python for Applied Machine Learning: supplementary material

Chris McCool

1 Overview

These notes serve as supplementary material. This is helpful for students taking the "Python for Applied Machine Learning" course. It is written from the point of view of applied machine learning while also containing some tips on how things can be implemented in Python.

With regards to the "Python for Applied Machine Learning" course, or PAML, there are several textbooks with other information that could be of help.

For Python and machine learning with Python (e.g. NumPy/SkLearn) the following are reference textbooks that are, or were, available as e-books through the University of Bonn library.

- "Introduction to Programming in Python: An Interdisciplinary Approach";
- "SciPy and NumPy"; and
- "Introduction to Machine Learning with Python".

For background into machine learning and pattern recognition ideas there are several other textbooks that are recommended.

- "Pattern Recognition and Machine Learning" by Christopher Bishop;
- "Pattern Classification" by Duda, Hart and Stork;
- "Bayesian Reasoning and Machine Learning" by David Barber; and
- "Kernel Methods for Pattern Analysis" by Shawe-Taylor and Cristianni.

2 Basics of Vector and Matrix Mathematics

Vector and matrix arithmetic are incredibly important. They allow us to represent very complex relationships in a compact form that is then also useful for providing derivations. Rather than going into the minutia of how vector and matrix operations and derivations work, we highlight their use in what we use. A good source for identities that are extremely useful for a range of problems the interested reader is pointed to an online text called "Matrix Cookbook" by Petersen and Pedersen.

In general, our data matrix $\mathbf{X}$ is of size (m, n). It consists of m rows and n columns. Each row corresponds to a sample and each column corresponds to a feature within a feature vector; sometimes called the dimensionality of the feature vector. When we have vectors in maths, we consider them to be a column-vector which has n rows and 1 column or size $(n, 1)$. By contrast a row-vector has 1 row and n columns or size $(1, n)$. In Python (NumPy), vectors are a special case that is neither a column-vector nor a row-vector, later we describe how this can lead to issues in NumPy if you attempt to directly implement certain equations. Vectors and matrices are usually indicated using the **bold typeface**. A vector is lower case ($\mathbf{w}$) and a matrix is uppercase ($\mathbf{W}$).

Below we describe some basic properties including the magnitude of a vector, the unit vector and transpose of vectors and matrices, diagonal matrices, covariance matrices and some notes on using vectors in NumPy.

2.1 Vector Magnitude and Unit Vectors

The magnitude of a vector $\mathbf{x}$ is the length of this vector and is given by

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}. \tag{1}$$

Writing the above equation when n becomes large is obviously cumbersome and so we can do this more compactly by writing

$$\|\mathbf{x}\| = \sqrt{\sum_{j=1}^{n} x_j^2}. \tag{2}$$

2.2 Transpose

The transpose operation swaps rows for columns and columns for rows. This can be applied to vectors and matrices and is given by the symbol T. The transpose of a column-vector is a row-vector is of size $(1, n)$, denoted by $\mathbf{a}^T$. The transpose of the matrix $\mathbf{X}$ of size (m, n) is $\mathbf{X}^T$ a matrix of size (n, m).

2.3 Diagonal and Identity Matrices

A diagonal matrix $\mathbf{D}$ is a matrix whose entries are all zero except on the diagonal elements. For a matrix of size $(3, 3)$ this is given by

$$\mathbf{D} = \begin{bmatrix} d_{1,1} & 0 & 0 \\ 0 & d_{2,2} & 0 \\ 0 & 0 & d_{3,3} \end{bmatrix} \tag{3}$$

The identity matrix, $\mathbf{I}$, is a special case of the diagonal matrix where the diagonal elements are all 1. For a matrix of size $(3, 3)$ this is given by

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

2.4 Covariance Matrix

A covariance matrix for vectors of size $(n, 1)$ is a matrix $\mathbf{\Sigma}$ of size (n, n). Covariance matrices are symmetric and this leads to the property that $\mathbf{\Sigma}^T = \mathbf{\Sigma}$. Usually a covariance matrix is calculated from mean-centered vectors. When calculating for m samples such as from the data matrix $\mathbf{X}$ we could write

$$\mathbf{S} = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i - \mu)^T (\mathbf{x}_i - \mu) \tag{5}$$

where μ is the mean of the samples in $\mathbf{X}$.

2.5 Python and Vectors

As this is oriented towards Python and we make use of matrices and vectors it is important to note that we use the NumPy library. This means that a Python vector is of size $(n,)$, this represents a vector in NumPy and it should be treated as such. Therefore, it is neither a row- nor column-vector but a Python vector. If you attempt to implement an equation assuming this is either a row- or column-vector this will lead to errors, should you wish to do that you will need to convert the Python vector to be a row-or column-vector. An example of how to do this is given below.

```
n = a.shape
r_vec = np.reshape(a,(1,n))  # A row vector (1 , n)
c_vec = np.reshape(a, (n,1)) # A column vector (n , 1)
```

3 Frequently Used Distances

There are numerous distances and similarity measures used in machine learning. Of these, there are some that occur frequently and will be used in this course and so we provide a more detailed breakdown of these below. Distances should produce small values when the vectors are similar and larger values when they are dissimilar. When the vectors are identical they should result in 0.

3.1 Euclidean Distance

The Euclidean distance is one of the most frequently used measures. It is also called the L_2 distance. We present two formulations for this that are equivalent. In the first formulation, we write the distance using column-vectors and it is given as

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b})} \tag{6}$$

The second equivalent formulation is

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{j=1}^{n} (a_j - b_j)^2} \tag{7}$$

An advantage of presenting this in terms of the first formulation is that makes it clearer the relationship between this distance and the Mahalanobis distance.

3.2 Sum of Square Differences

The Euclidean distance includes a square root in its formulation. Sometimes, this is dropped in which case it is usually referred to as the sum of squared differences

$$d(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^{n} (a_j - b_j)^2 = (\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b}). \tag{8}$$

The sum of squared differences is also used as the starting point to get several objective functions.

3.3 Mahalanobis Distance

The mahalanobis distance has many settings and discussing all of these is beyond the scope of this document. It can be used as the distance between a vector $\mathbf{x}$ and a distribution given by $\boldsymbol{\Theta}$. Often the parameters of the distribution are a mean vector (μ) and a covariance matrix ($\boldsymbol{\Sigma}$) such that $\boldsymbol{\Theta} = [\mu, \boldsymbol{\Sigma}]$. In this case, the distance can be expressed as

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a} - \mathbf{b})^T \boldsymbol{\Sigma}^{-1} (\mathbf{a} - \mathbf{b})}. \tag{9}$$

This has similarities to the Euclidean distance and is equivalent in the case that the covariance matrix is the identity matrix $\mathbf{I}$.

3.4 Cosine Similarity

The cosine or angular similarity measures the angular similarity. This is related to the dot product

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta. \tag{10}$$

The cosine similarity is given by

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}. \tag{11}$$

Or more simply it can be written as

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}. \tag{12}$$

Noting that the range of values is $[-1, 1]$ this can be changed to be a distance of range $[0, 1]$ by doing the following

$$d(\mathbf{a}, \mathbf{b}) = \frac{\text{sim}(\mathbf{a}, \mathbf{b}) + 1}{2}. \tag{13}$$

In full this becomes

$$d(\mathbf{a}, \mathbf{b}) = \left(1 + \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}\right) / 2. \tag{14}$$

3.4.1 Vector Dot Product

The vector dot product $(\cdot)$ relates to the cosine similarity and is also called the inner product. We saw its definition earlier but we also write it separately and clearly here to be

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}. \tag{15}$$

This can also be written as

$$\mathbf{a} \cdot \mathbf{b} = \sum_{j=1}^{n} a_j b_j. \tag{16}$$

3.5 Gaussian or Normal Distribution

One of the most common distributions is the normal distribution also called the Gaussian function. For a single feature (variable), it is defined by two parameters ($\boldsymbol{\Theta}$) the mean μ and variance σ^2. The likelihood that a sample x was produced by a Gaussian function with parameters $\boldsymbol{\Theta}$ is

$$p\left(x \mid \boldsymbol{\Theta}\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{2}\right)^2}. \tag{17}$$

For multi-variate data, more than one feature (e.g. feature vectors), the parameters are the mean vector μ and the covariance matrix $\boldsymbol{\Sigma}$. This leads to the likelihood that a sample $\mathbf{x}$ was produced by a Gaussian function with parameters $\boldsymbol{\Theta}$ is

$$p\left(\mathbf{x} \mid \boldsymbol{\Theta}\right) = (2\pi)^{-\frac{n}{2}} \det \boldsymbol{\Sigma}^{-\frac{1}{2}} \exp^{-\frac{1}{2}(\mathbf{x}-\mu)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mu)} \tag{18}$$

The only part of this equation which depends on the input feature vector $\mathbf{x}$ is the quadratic part in the exponential

$$\left(\mathbf{x} - \mu\right)^T \boldsymbol{\Sigma}^{-1} \left(\mathbf{x} - \mu\right) \tag{19}$$

This has similarities with the previously described Mahalanobis distance. Note that because this is a distribution it will calculate a likelihood.

4 Useful Derivative and Properties

There are some commonly occuring properties and identities that we use. We present some of these below. Many of these have been obtained and to some extent re-worked from the "Matrix Cookbook".

4.1 Derivatives

The following two derivatives assume that $\mathbf{W}$ is a symmetric matrix, this is the case for covariance matrices.

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{x}-\mathbf{a})^T \mathbf{W} (\mathbf{x}-\mathbf{a}) = 2\mathbf{W} (\mathbf{x}-\mathbf{a}) \tag{20}$$

$$\frac{\partial}{\partial \mathbf{a}} (\mathbf{x}-\mathbf{a})^T \mathbf{W} (\mathbf{x}-\mathbf{a}) = -2\mathbf{W} (\mathbf{x}-\mathbf{a}) \tag{21}$$

In addition to the above, when $\mathbf{W} = \mathbf{I}$ we have

$$\frac{\partial}{\partial \mathbf{a}} (\mathbf{x}-\mathbf{a})^T \mathbf{W} (\mathbf{x}-\mathbf{a}) = -2 (\mathbf{x}-\mathbf{a}) \tag{22}$$

Another useful derivative is the following

$$\frac{\partial}{\partial \mathbf{a}} (\mathbf{W}\mathbf{a}-\mathbf{x})^T (\mathbf{W}\mathbf{a}-\mathbf{x}) = 2\mathbf{W}^T (\mathbf{S}\mathbf{a}-\mathbf{x}). \tag{23}$$

4.2 Exponentials and Logarithms

The log of an exponential is the exponent of the exponential

$$\log [\exp (\mathbf{x})] = \mathbf{x}. \tag{24}$$

The log of a set of products is the sum of the logs.

$$\log \left[\prod_{i=1}^{m} \mathbf{x}_i \right] = \sum_{i=1}^{m} \log [\mathbf{x}_i]. \tag{25}$$

Where the product is

$$\prod_{i=1}^{m} \mathbf{x}_i = \mathbf{x}_1 * \mathbf{x}_2 * \cdots * \mathbf{x}_m \tag{26}$$

and the sum is

$$\sum_{i=1}^{m} \mathbf{x}_i = \mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_m. \tag{27}$$

5 Linear Regression

In linear regression we can cast the problem as we want to have a predictor function, $h(\mathbf{x} \mid \mathbf{\Theta})$, such that it provides the regressed (desired) output. It does this by transforming the input using a weighted linear combination of the features (linear function). In this case we will be writing that our parameters $\mathbf{\Theta} = \mathbf{s}$ are a vector consisting of the weights and so we can write

$$h(\mathbf{x} \mid \mathbf{s}) = s_0 + x_1 s_1 + x_2 s_2 + \cdots + + x_n s_n. \tag{28}$$

The value s_0 is the bias. We can put this together in matrix form by adding to our feature vectors, $\mathbf{x}$, an extra element of value 1 which is used to include the bias term s_0 so our new vector becomes

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{29}$$

This assumes we can summarise our parameters as

$$\mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}. \tag{30}$$

Using the above we can now write that for a single sample $\mathbf{x}$

$$h(\mathbf{x} \mid \mathbf{s}) = \mathbf{x}^T \mathbf{s}. \tag{31}$$

And if we wanted to write this for multiple samples, such as all of our training set, we could write

$$h(\mathbf{X} \mid \mathbf{s}) = \mathbf{X}\mathbf{s}. \tag{32}$$

Fundamentally, we want to minimise the error in our model. If we use the sum of square differences to do this then we can write that we want the sum of square difference between our labels $\mathbf{y}$ and our model $h(\mathbf{X} \mid \mathbf{s})$ to be minimised

$$J = \left[h(\mathbf{X} \mid \mathbf{s}) - \mathbf{y}\right]^T \left[h(\mathbf{X} \mid \mathbf{s}) - \mathbf{y}\right] \tag{33}$$

which can also be written as

$$J = (\mathbf{X}\mathbf{s} - \mathbf{y})^T (\mathbf{X}\mathbf{s} - \mathbf{y}). \tag{34}$$

This is an example of an objective function J that we want to minimise.

To minimise the above objective function, we need to differentiate with respect to the parameters and set the result to 0

$$\frac{\partial J}{\partial \mathbf{s}} = \frac{\partial}{\partial \mathbf{s}} (\mathbf{X}\mathbf{s} - \mathbf{y})^T (\mathbf{X}\mathbf{s} - \mathbf{y}). \tag{35}$$

Using the previously defined identity, (23), gives us that

$$\frac{\partial J}{\partial \mathbf{s}} = 2\mathbf{X}^T (\mathbf{X}\mathbf{s} - \mathbf{y}). \tag{36}$$

Setting the left hand side to 0 gives us

$$0 = 2\mathbf{X}^T (\mathbf{X}\mathbf{s} - \mathbf{y}) = 2\mathbf{X}^T\mathbf{X}\mathbf{s} - 2\mathbf{X}^T\mathbf{y}. \tag{37}$$

Then dividing both sides by 2 we get

$$0 = \mathbf{X}^T\mathbf{X}\mathbf{s} - \mathbf{X}^T\mathbf{y} \tag{38}$$

and rearranging the terms leads to

$$\mathbf{X}^T\mathbf{X}\mathbf{s} = \mathbf{X}^T\mathbf{y}. \tag{39}$$

Finally we substitute $\mathbf{z} = \mathbf{X}^T\mathbf{y}$ which is a vector of size $(n, 1)$ and $\mathbf{W} = \mathbf{X}^T\mathbf{X}$ is a matrix of size (n, n) gives us

$$\mathbf{W}\mathbf{s} = \mathbf{z}. \tag{40}$$

The above can be solved in two ways:

1. we could take the inverse of $\mathbf{W}$ to get $\mathbf{s} = \mathbf{W}^{-1}\mathbf{z}$, or

2. as it has the form of a system of linear equations $\mathbf{Ws} = \mathbf{z}$ use solvers for this.

The reason we would take the second option is that it is more numerically stable as calculating the inverse of a matrix $\mathbf{W}^{-1}$ can introduce imprecision. The general way to write the system of linear equations is $\mathbf{Ax} = \mathbf{b}$. The code for doing this in Python is as follows

```
import numpy as np
s = np.linalg.solve(W, z)
```

6 Implementing Equations in Python

The following is a brief guide to implementing or using some of the approaches discussed above in Python. Pseudo-code has been provided to attempt to further elaborate on this. First, Python considers vectors to be of size $(n,)$. This is usually efficient and usually not a problem unless you try to directly implement an equation. If you do want to directly implement an equation the suggestion is to use to the Python vectors to be column vectors using the reshape command. An example of this for the Python vector "a" is

```
n = a.shape
r_vec = np.reshape(a,(1,n))  # A row vector (1 , n)
c_vec = np.reshape(a, (n,1)) # A column vector (n , 1)
```

6.1 Dot Product a and b: $\mathbf{a}^T\mathbf{b}$

The equation $\mathbf{a}^T\mathbf{b}$ is also the dot product between two vectors $\mathbf{a}$ and $\mathbf{b}$. It is the sum of the products of two vectors

$$\sum_{j=1}^{n} a_j b_j. \tag{41}$$

To achieve this in Python you could write for two Python vectors $(n,)$

```
z = a*b # Perform the multiplication element-wise
dot_product = z.um() # Sum up the resultant vector
```

For this example we will explain a little bit more. The code for "a*b" results in the vector "z" of size $(n,)$ that contains the multiplication each element from a and b such that the i-th element of z is given by

$$z_i = a_i * b_i. \tag{42}$$

Taking the sum of this vector then gives the desired result of the dot product. A short-hand way of writing this Python code is given by

```
dot_product = z.um() # Sum up the resultant vector
```

Note: the "*" operation in Python is an element-wise multiplication.

6.2 Sum of squared differences: $(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})$

The quadratic $(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})$ is something we will see frequently. This also results in the sum of squared differences seen in the L_2 distance. Let's write this out more fully by first setting $\mathbf{z} = \mathbf{a}-\mathbf{b}$ and so this becomes

$$\mathbf{z}^T\mathbf{z} = \sum_{j=1}^{n} z_j z_j \tag{43}$$

We saw earlier an example of this, we can further simplify for this example as it is the squared value of the differences and then a summation as

```
diff = a-b # Of size (n,)
square_diff = diff**2 # Of size (n,)
sum_square_diff = square_diff.sum() # A floating point value
```

Or all in one line as

```
sum_square_diff = ((a-b)**2).sum() # A floating point value
```