International e-Conference on New Horizons And Multidisciplinary Applications In Science And Technology
In Association withInternational Journal of Scientific Research in Science and Technology
Volume 9 | Issue 6 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)

Twitter Based Sentiment Analysis - Theoretical Aspects

Rajendra T Kaple

The model

Logistic regression is a method for analyzing a dataset in which there are one or more independent variables that determine an outcome (which is binary). In logistic regression, the dependent variable is binary i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or a 0 (FALSE, failure, non-pregnant, etc.). The goal of logistic regression is to find the best fitting model to describe the relationship between the dependent variable (which is binary) and a set of independent variables.

Consider a scenario where we need to classify whether an email is spam or not. So we set a threshold 0.5 any value above 0.5 will consider True and below 0.5 will be False.

Types of Logistic Regression

1. Binary Logistic Regression

The categorical response has only two 2 possible outcomes. Example: Spam or Not

2. Multinomial Logistic Regression

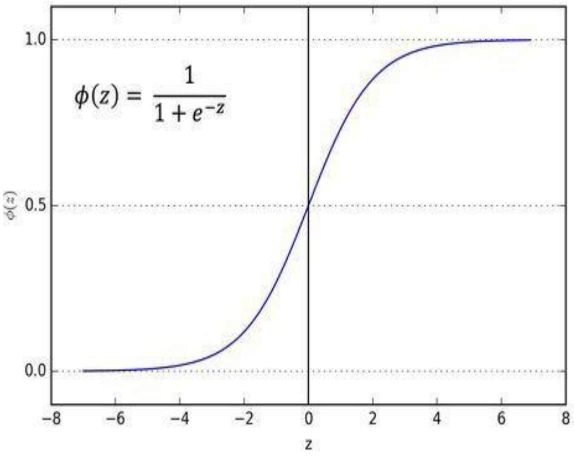
Three or more categories without ordering. Example: Predicting which food is preferred more (Veg and Non-Veg, Vegan)

3. Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 5.

A. Logistic Function

The logistic function, also called the sigmoid function It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.



Decision Boundary:

To predict which class a data belongs, a threshold can be set. Based upon this threshold, the classes are divided Say, if predicted_value ≥ 0.5, then classify email as spam else as not spam.

Decision boundary can be linear or nonlinear. Polynomial order can be increased to get complex

Cost Function

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

Why cost function which has been used for linear cannot be used for logistics?

Linear regression uses mean squared error as its cost function. If this is used for logistic regression, then it will be a non-convex function of parameters (theta), and as we all know Gradient descent converge into global minimum only if the function is convex.

Cost function explanation

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$$\text{Note: } y = 0 \text{ or } 1 \text{ always}$$

We know that there are only two possible cases

- y = 1

Then our equation simplifies to

- ■ -log(h θ (x)) - (0)log(1 - h θ (x))
    - ● -log(h θ (x))

- ○ y = 1
- ○ Then our equation simplifies to
    - ■ -(0)log(h θ (x)) - (1)log(1 - h θ (x))
    - ■ = -log(1- h θ (x))

Why do we choose this function when other cost functions exist?

We use this cost function because it is used to find the probability of an event = success or event = failure

Here we use binomial distribution, logistic regression does not require linear relation between dependent and independent variable. It can handle various types of relationships because it applies a non-linear log transformation to predict (odd ratio) values.

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

Odds=p/(1-p)

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}$$

and

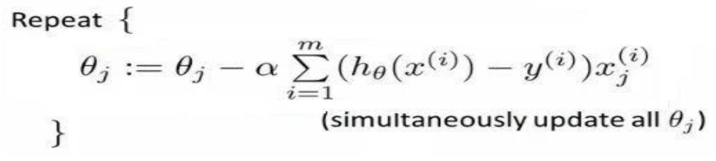
$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression) estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values.

Gradient Descent

Now we need to figure out how to minimize J(θ).

Use gradient descent as before Repeatedly update each parameter using a learning rate



Repeat {

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

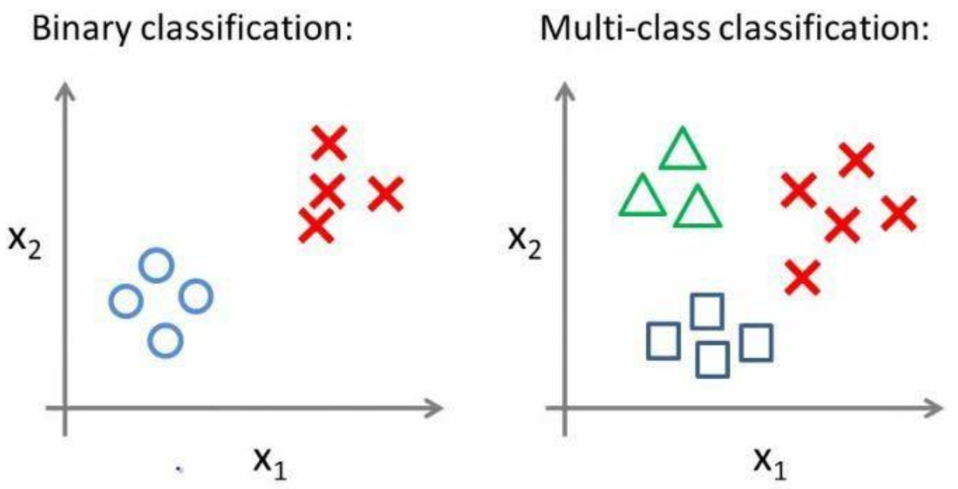
(simultaneously update all θ_j)

}

If you had n features, you would have an n+1 column vector for θ. This equation is the same as the linear regression rule

Multivariate Logistic Regression

Multivariate logistic regression is a classification method for multiclass(more than two possible discrete outcomes) problems. It is a model that is used to predict the probabilities of the different possible outcomes of Classes.



Multi class classification is implemented by training multiple logistic regression classifiers, one for each of the N classes in the training dataset.

In the above Multi Class classification example, there are 3 classes. Hence, we need to train 3 different logistic regression classifiers.

When training the classifier for Class 1, we will treat input data with class 1 labels as +ve samples (y=1) and all other classes as -ve samples (y=0).

When training the classifier for Class 2, we will treat input data with class 2 labels as +ve samples (y=1) and all other classes as -ve samples (y=0).

This will continue for all the classes.

A. Prediction phase: One vs all prediction

Once training all our classifiers, we can now use it to predict which class the test data belongs to.

For the test input, we should check the "probability" that it belongs to each class using the trained logistic regression . Your one-vs-all prediction function will pick the class for which the corresponding logistic regression outputs the highest probability and return the class label (1,2,..., or N) as the prediction for the input example.

The terms that we are going to be used frequently in this paper

- **Kernel:** The function used to map a lower dimensional data into a higher dimensional data.
- **HyperPlane:** In SVM this is basically the separation line between the data classes. Although in SVR we are going to define it as the line that will will help us predict the continuous value or target value
- **Boundary line:** In SVM there are two lines other than Hyper Plane which creates a margin . The support vectors can be on the Boundary lines or outside it. This boundary line separates the two classes. In SVR the concept is same.
- **Support vectors:** This are the data points which are closest to the boundary. The distance of the points is minimum or least.

B. Supervised Virtual Machine (SVM) :

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs. Simply put, it does some extremely complex data transformations, then figures out how to separate your data based on the labels or outputs you've defined.

Here we take a point from each cluster which is the farthest from mean and from the boundary using those points. Suppose you are given plot of two label classes on graph as shown in image (A). Can you decide a separating line for the classes?

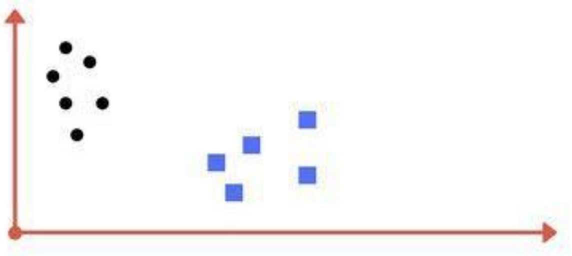


Image A: Draw a line that separates black circles and blue squares.

You might have come up with something similar to following image (image B). It fairly separates the two classes. Any point that is left of line falls into black circle class and on right falls into blue square class. Separation of classes. That’s what SVM does. It finds out a line/ hyper-plane (in multidimensional space that separate outs classes). Shortly, we shall discuss why I wrote multidimensional space.

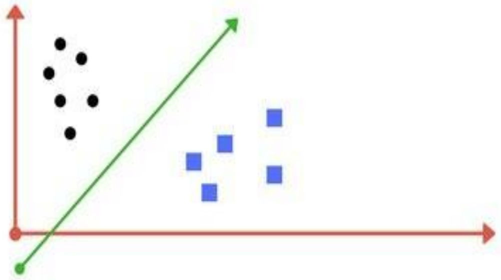
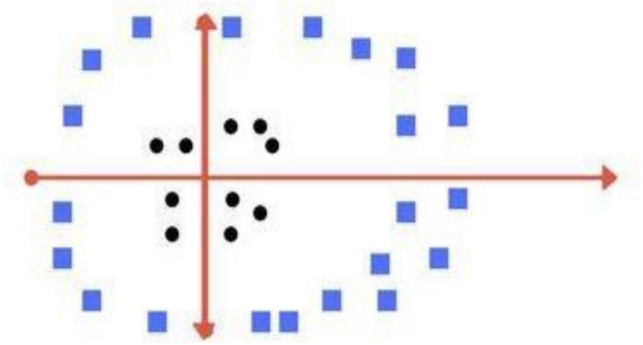
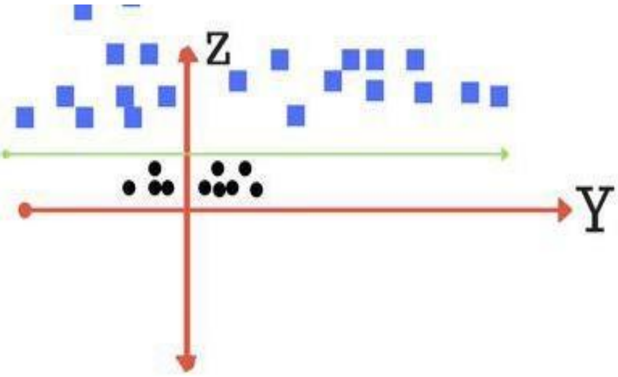


Image B: Sample cut to divide into two classes.

Now consider what if we had data as shown in image below? Clearly, there is no line that can separate the two classes in this x-y plane. So what do we do? We apply transformation and add one more dimension as we call it z-axis. Let's assume value of points on z plane, $w = x^2 + y^2$. In this case we can manipulate it as distance of point from z-origin. Now if we plot in z-axis, a clear separation is visible and a line can be drawn

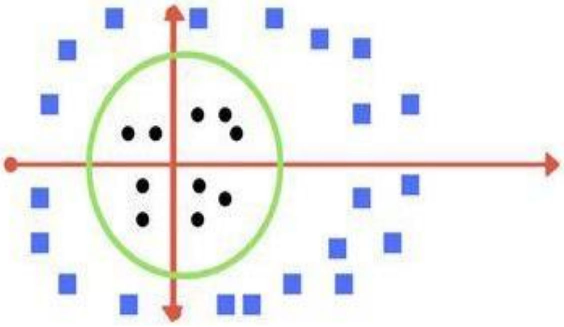


Now can you draw a separating line in this plane?



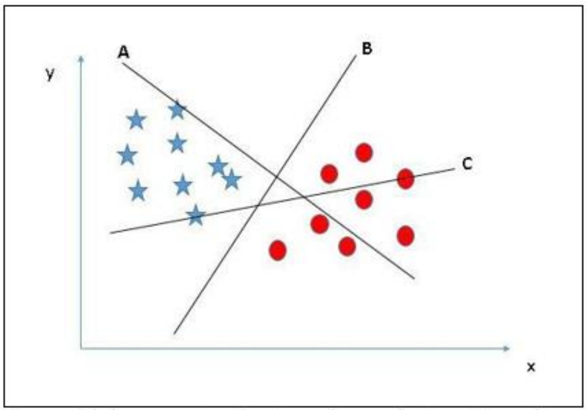
Plot of zy axis. A separation can be made here.

When we transform back this line to original plane, it maps to circular boundary as shown in image E. These transformations are called kernels.



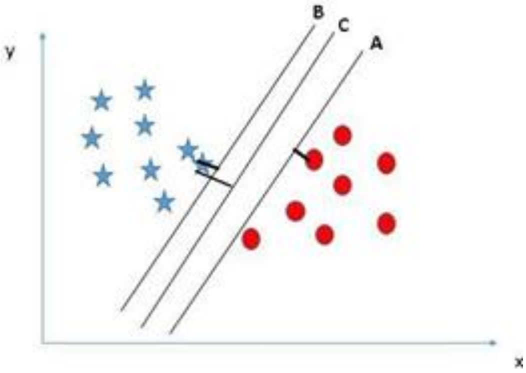
Transforming back to x-y plane, a line transforms to circle.

Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.

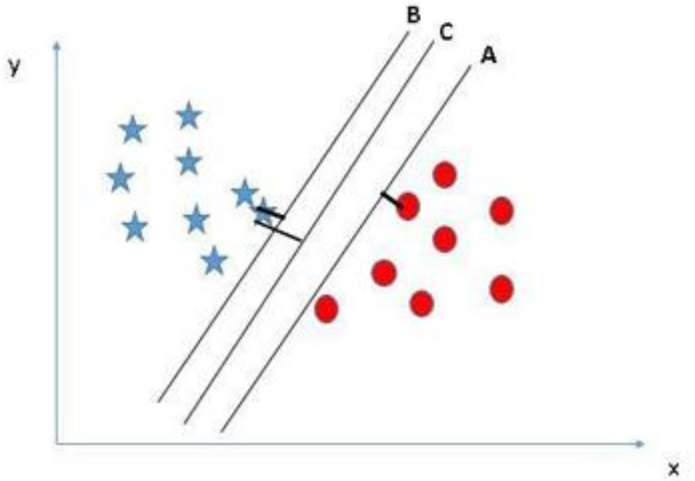


Select the hyper plane which separates the two classes better. Here, hyper-plane “B” separates the two classes better.

- Here, we have three hyper-planes (A, B and C) and all are separating the classes well. Now, How can we identify the right hyper-plane?



Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as Margin. Let’s look at the below snapshot:



In this figure we can see that line C has high margin than A and B. So, line C is the right hyper plane in this situation. Another reason to select the hyperplane with higher margin is robustness and to avoid misclassification. SVM has feature to ignore outliers. Hence we can say that it’s robust.

E. Pros and Cons associated with SVM

● Pros:

 It works really well with clear margin of separation

 It is effective in high dimensional spaces.

 It is effective in cases where number of dimensions is greater than the number of samples.

 It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

● Cons:

 It doesn’t perform well, when we have large data set because the required training time is higher.

 It also doesn’t perform very well, when the data set has more noise i.e. target classes are overlapping.

 SVM doesn’t directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python
scikit-learn library.

F. Linear Regression :

There are two types of Linear regression :

a. Simple Linear Regression :

When we have only one feature, then it comes under univariate linear regression

b. Multi-Linear Regression :

When we have more than one feature, then it comes under simple linear regression. In linear regression, we are going to find the best fit line for our training dataset. So, what is a best-fit line? Best fit line is the one where we get the minimum difference between predicted values and actual values, we have few mathematical formulae to find the best line and now we are going to see how to use them, but first few terminologies:

1. Hypothesis Function :

- Take the training data set
- Pass it to the machine learning algorithm
- It gives a function as output(denoted by h). This function is called Hypothesis

function.

➔ Here Θi represents the parameters.

➔ θ0 is a point where slope intercepts y-axis(or constant)

➔ θ1 is slope(or gradient)

- Now pass, test data set to the hypothesis function and it outputs the prediction.

2. Loss Function: The loss function is the sum of the squared distance between predicted value and actual value of the label for a single observation.

3. Cost: Cost function is the average of summation of the loss function. It is denoted by J(θ0 , θ1).

4. Gradient Descent: It is a general algorithm that is used to minimize a function, in this case, it's Mean Squared Error Cost Function.

G. Linear regression - implementation (cost function) :

Loss Function: The loss function is the sum of the squared distance between predicted value and actual value of the label for a single observation. By squaring, the loss function amplifies the influence of bad predictions, i.e. it reacts more strongly to outliers.

The formula for loss function

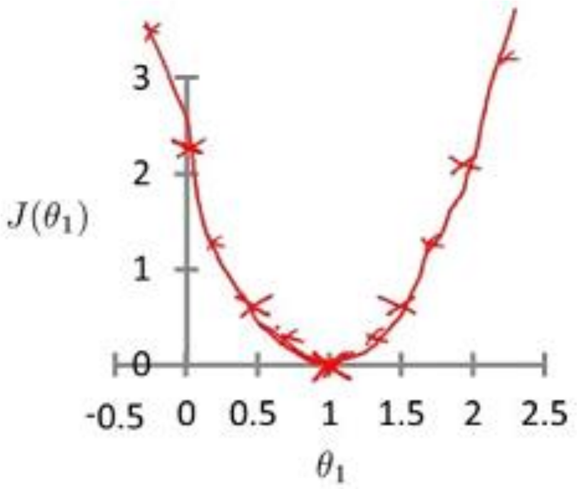
Cost: Cost function is the average of summation of the loss function. It is denoted by J(θ0 , θ1).

We are dividing the cost function by 2 only to reduce its value, just to make further calculation easy and small. It determines how our hypothesis behaves for different value for parameters. The cost is higher when the model is performing poorly on the training set.

So, why do we take the square of error?

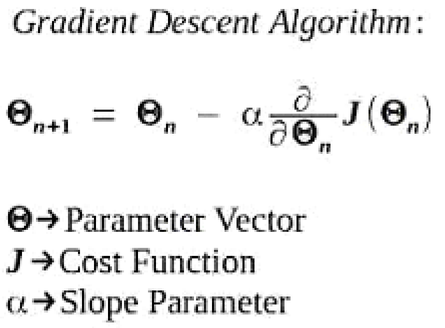
Error finds the difference between predicted and original value. We are doing this using linear regression. So, mathematically we are finding the distance between two points and we all know that distance can never be negative and hence we are squaring it.

When we plot the graph by joining the points generated by loss function we get a convex curve.



The optimization objective for the learning algorithm finds the value of θ1 which

Minimizes J(θ1)



Gradient Descent Algorithm:

$$\Theta_{n+1} = \Theta_n - \alpha \frac{\partial}{\partial \Theta_n} J(\Theta_n)$$

Θ→Parameter Vector

J→Cost Function

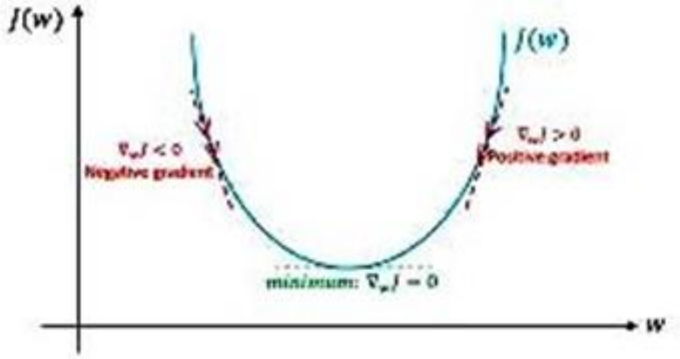
α→Slope Parameter

Why do we take the Derivative of Cost Function? :

There are two reasons for taking the derivative :

1. To find the direction we want to move so that we'll get a minimum of cost function :

In the plot of cost function vs theta, a positive slope means function moves upward as we move right, so we have to move towards the left to get the minimum value of cost function.



Similarly, a negative slope means function moves downwards as we move left, so move towards the right to get the minimum value of cost function.

2. How big the step should be, so that we won't miss the global minima (the point where the value of cost function is minimum) :

Here we need to take larger steps if the slope is larger i.e., we are so far from the minimum and smaller steps if the slope is smaller.

Learning Rate : (alpha)

To control the size of the steps we use learning rate. Smaller the learning rate, smaller will be the steps. Selecting the learning rate is one of the critical tasks because it affects the time for training the program. When the learning rate is large, we might miss the minimum value. If the learning rate is small, we might have to run a number of iterations, which in turn increases the training time.

So, how can we decide how many iterations to take so as to get suitable values of parameters?

Epochs :

We need to iterate the gradient descent a certain number of times to obtain the suitable values of parameters. For this, we have to go for try and error method. So, here we need to calculate the certain large number of times.

```
Repeat until convergence {
    $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
}
```

The intuition of cost function and gradient descent :

Here, basically, we have to find the values of parameters where we get the minimum value of loss function. For doing so, we have to go for try and error method, i.e., we have to calculate loss function for different values of theta_0 and theta_1. And this might require thousands of attempts. We need to find a value of J(theta) after which whatever the value of theta might be, loss function won't change much.
This process of finding all the values of parameters is cost function and finding values of parameters (theta_0 and theta_1) where the value of cost function(J(theta)) is minimum is

nothing but gradient descent. But using gradient descent we can find minimum value easily without storing all the values for parameters. Using const function along with gradient descent is termed as linear regression.

H. Gradient descent algorithm :

Derivation of GD :

Here features and label as constats. Here we are taking derivation on both sides of the equation of cost function.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad [1.0]$$

$$\frac{d}{d\theta_0}J(\theta_0, \theta_1) = \frac{d}{d\theta_0}\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right) \qquad [1.1]$$

$$= \frac{1}{m}\sum_{i=1}^{m}\frac{d}{d\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad [1.2]$$

$$= \frac{1}{m}\sum_{i=1}^{m}2\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\frac{d}{d\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \qquad [1.3]$$

$$= \frac{2}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \qquad [1.4]$$

To move from equation [1.1] to [1.2], we need to apply two basic derivative rules:

Scalar multiple rule: $\frac{d}{dx}(\alpha u) = \alpha \frac{du}{dx}$

Sum rule: $\frac{d}{dx}\sum u = \sum \frac{du}{dx}$

Moving from [1.2] tog [1.3], we apply both the power rule and the chain rule:

Power Rule $\frac{d}{dx} u^{n} = n \quad u^{n-1} \frac{du}{dx}$

Finally, to go from [1.3] to [1.4], we must evaluate the partial derivative as follows. While taking the partial derivative we have to keep in mind that everything except Θ0 is constant, i.e., Θ1, x, and y are constants here

Chain Rule $\frac{d}{dx} f(g(x)) = f'(g(x))\, g'(x)$

$$\frac{d}{d\theta_0} \quad (h_\theta(x^{(i)}) - y^{(i)}) \quad = \quad \frac{d}{d\theta_0} \quad (\theta_0 + \theta_1 x^{(i)} - y^{(i)})_{=} \mathbf{1}$$

Equation [1.4] gives us the partial derivative of the MSE cost function with respect to one of the variables, Θ0. Now we must also take the partial derivative of the MSE function (cost function) with respect to Θ1. The only difference is in the final step, where we take the partial derivative of the error:

Multiplying the cost function by a scalar does not affect the location of its minimum, so we can get away with this. So, here we divide the equation by 2 just to reduce it so that the calculations get simple. So as a final result we get

Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Objective:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Update rules:

$$\theta_0 := \theta_0 - \alpha \frac{d}{d\theta_0} J(\theta_0, \theta_1)$$
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

In multi linear regression, we have more than one features. So just like linear regression we will have first parameter with no "X" variable and rest all will have "X" value i.e., their corresponding feature with them like

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

$\cdots$

}

this where X_0 = 1 :

We can combine and write the equation for multi linear gradient descent as : for i in range(len(traindata)) :
for j in range(len(traindata)) :

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

Where ,

i - row(observation) j - column(feature)

Example of simple linear regression :

In linear regression, we need to generate a straight line(aka regression line) where errors are minimized.

Directory Structure of the Problems and naming conventions of the files :

Here I am taking World happiness for example

1. World Happiness Prediction Problem (folder)

a. Model Training (folder)

i. Data (folder)

1. world_happiness_training_dataset.csv (csv file)

ii. Model (folder)

1. world_happiness_training.pkl (pickle file)

iii. world_happiness_trainig.py (python file)

iv. world_happiness_trainig.ipynb (jupyter python file)

v. CSVs Files (folder)

b. Model Testing (folder)

i. Data (folder)

1. world_happiness_testing_dataset.csv (csv file)

ii. world_happiness_testing.pkl (pickle file)

iii. world_happiness_testing.py (python file)

iv. world_happiness_testing.ipynb (jupyter python file)

c. requirement.txt (text file)

Please change the above problem name with your problem name.

I. Environment Setup:

Step 0: Check the python 3 installed or not

● Type python3 -V cmd on the terminal it will show you it is installed or not

○ If not installed then install the python using the following commands

■ sudo apt-get update

■ sudo apt-get upgrade python3

■ sudo apt-get install python3

Step 1:

● Install pip first

○ sudo apt-get install python3-pip

● Then install virtualenv using pip3

○ sudo pip3 install virtualenv

● Install Jupyter

○ pip3 install jupyter

Step 2:

● Clone the problems from Github repository

○ git clone https://github.com/AkankshaKaple/ML_Problems

● Unzip the directory

● Switch to the ML_Problems directory

● Now create a virtual environment

○ python3 -m venv myenv

● Active your virtual environment:

○ source myvenv/bin/activate

● Install the required libraries and packages

○ pip3 install -r requirements.txt (this file you get in the cloned repo)

● Run Jupyter Notebook in ML_Problems directory

○ jupyter notebook

● Set different kernel in Jupyter

○ python3 -m ipykernel install --user --name myenv --display-name "myenv".

Machine Learning Terms Libraries :

● Pandas : In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis and storing in a proper way. In particular, it offers data structures and operations for manipulating numerical tables and time series.

● Numpy : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

● Pickle : Python pickle module is used for serializing and de-serializing a Python object structure. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to
re-construct the object in another python script.

● Sklearn : Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn.
● NLTK : The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.
● Seaborn : Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures. Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data.

Natural Language Processing (NLP) Terms :

● Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.
● Lemmatization:
Layers in Keras:
● Dense: In simple words, a dense layer is where each unit or neuron is connected to each neuron in the next layer.
● Dropout :
 Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting.
 Using “dropout", you randomly deactivate certain units (neurons) in a layer with a certain probability p from a Bernoulli distribution (typically 50%, but this yet another hyper parameter to be tuned). So, if you set half of the activations of a layer to zero, the neural network won’t be able to rely on particular activations in a given feed-forward pass during training. As a consequence, the neural network will learn different, redundant representations; the network can’t rely on the particular neurons and the combination (or interaction) of these to be present. Another nice side effect is that the training will be faster.
 Dropout is only applied during training, and you need to rescale the remaining neuron activations. E.g., if you set 50% of the activations in a given layer to zero, you need to scale up the remaining ones by a factor of 2. Finally, if the training has finished, you’d use the complete network for testing (or in other words, you set the dropout probability to 0).

Twitter Analysis :
● Anything to add from the document: https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-data set-python/

● When to use a bag of words:
If your dataset is small and context is domain-specific, Basic operational Weight (BoW) may work better than Word Embedding. Context is very domain-specific which means that you cannot find the corresponding Vector from pre-trained word embedding models (GloVe, fastText etc).

● Random State in train_test_split() :

When we use a random number generator for number/sequence generation, we give a starting number (AKA seed). When we provide the same seed, every time it'll generate the same sequence as the first one. That's why to keep the same random values every time, we give seed as random_state in train_test_split().

● Random Number:

The random numbers which we call are actually "pseudo-random numbers". These are generated by some kinds of deterministic algorithms. Basically, these pseudo-random numbers follow some kinds of sequences which has very very large period . That's why it appears random to us but in fact, are patterns in the
long run.

● What is a seed?

□ The seed during such random number generation is actually the starting point in the sequence. If we use the same seed every time, it will yield the same sequence of random numbers.

□ So, the same seed yields the same sequence of random numbers.

□ The reason for using a seed of some value is when we want to debug the program using such deterministic behavior.

Advantages of Adam:

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based on training data.

When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows:

● Straight forward to implement.
● Computationally efficient.
● Little memory requirements.
● Invariant to diagonal rescale of the gradients.
● Well suited for problems that are large in terms of data and/or parameters.
● Appropriate for non-stationary objectives.
● Appropriate for problems with very noisy/or sparse gradients.
● Hyper-parameters have intuitive interpretation and typically require little tuning.

Algorithms :

● There is no transform in logistic regression and SVM.

Issues :

● Model selection prepro not working in age detection.