Mathematics for Machine Learning

Aditya Reddy, Claire Guo, and Cole Wasserman

Final Project for Discrete Mathematics

New York University
Professor Normand
December 2020

Contents

1 Abstract 2

2 Main Section 2

3 Proofs 6

4 Conclusion 10

5 References 10

1 Abstract

Machine learning is the process of mimicking human-like intuition through the capabilities of computer algorithms and the ability for machines to improve themselves through experience. Its applications are demonstrated through things such as data modeling, image/speech recognition, web recommendations/advertisements, etc. Unlike humans, who are able to see a given problem and instinctively produce a solution or method of solving the problem using our previous knowledge of what has worked and what has not, machines don't have this capability; instead, they are only able to do exactly as dictated in the algorithms that we provide them with. However, the benefit of this is that machines are significantly faster at performing repeated tasks than humans are. Take the problem of finding the sum of $1 + 2 + 3 + \cdots + 100$ for example. While there is obviously a formula to help us solve this problem faster, even applying it may take us a minute or so. A machine, however, given the correct code, would be able to find the solution in a fraction of a second, even by the method of iterating through each integer. Although that application may seem simple, the capacities of machines expand way beyond just solving addition problems. Machine learning takes the computer's ability to perform repetitive tasks quickly and translates it into a means to find patterns. The types of learning can be broken down into three categories: supervised learning involves providing the machine with the correct patterns to look for, unsupervised learning involves the machine finding whatever patterns there are, and reinforcement learning involves having a clear objective and trying out different things until the objective is met. In the main section, we will go into more detail about each of these.

These learning methods all involve elements of math (multivariable calculus and linear algebra, most frequently), and in this report we will explore their applications in two fundamental concepts of machine learning: cost functions and gradient descent.

2 Main Section

As mentioned earlier, there are three main types of machine learning algorithms: supervised learning, unsupervised learning, and reinforcement learning.

1. **Supervised learning**
   In supervised learning, the model is told the "correct" information and pattern to look for in the data that it is given. This is done by feeding the algorithm a list of input values that are paired with a distinct, correct output value and giving the algorithm a type of relationship to look for in the data. Typically, input data for supervised learning algorithms are split up into two categories, training and testing data. The training data, as the name suggests, is used to train and build the model on. Testing data, on the other hand is used to check that the model is making accurate predictions and not simply "memorizing" the data. Supervised learning has two main types of algorithms: classification and regression. Classification is the process of finding a function that maps the input values to a discrete output. Many classification algorithms would have outputs such as "yes/no". For instance, a classification algorithm could be used to predict whether a patient does or does not have a type of cancer based on the an input of symptoms that they report. Regression algorithms are used to predict a continuous output. For example, in linear regression, one of the simplest types of supervised learning algorithms, the algorithm will attempt to find a linear relationship between in data and make an equation of a line that best fits this data. This line can be used to make predictions about a certain type of data. A good problem that linear regression

would be useful in solving is attempting to predict a house price in a neighborhood given the house's square footage. Supervised learning, linear regression in particular, is going to be the main subject of this research paper.

2. **Unsupervised learning**
   Unsupervised learning is a notch up in terms of complexity from supervised learning. In unsupervised learning, the algorithm is not given an explicit pattern or instructions to look for in the data that it is given. It is the job of the algorithm to find a structure in the data without being told what to look for. Examples of an unsupervised learning algorithms are all clustering models such as k nearest neighbors clustering and hierarchical clustering.

3. **Reinforcement learning**
   Reinforcement learning is the most recent, complex, and commonly used process of learning, where the machine improves by trial and error. Similar to reinforcement learning in humans, the machine is rewarded or penalized for for its actions based on whether or not they were successful. An example of this learning method would be teaching a machine to master a game such as chess. Over time and through many trials, the machine would be able to learn and recall with moves/series of moves results in wins and losses, with advancement in the game being the reward and detriment in the game being the penalty. The greatest challenge associated with reinforcement learning is building the simulation environment. While simulating a game of chess may be rather straightforward, creating an environment to test self driving machines, for example, would likely not be as simple as other factors (e.g. cost, safety, available space, resources) become relevant.

Linear Regression
The most basic and standard concept involved in the understanding of machine learning algorithms, linear regression is the way of modeling a group of data through a linear representation, or in other words: $y = ax + b$. The process involves finding a linear model such that the sum of the differences between the actual values that are greater than their predicted values and the predicted values is roughly equal to the sum of the differences between the predicted values and the actual values that are less than their predicted values.

Example with pictures on a random set of points:

Cost Function
Used to maximize accuracy, this describes how closely a linear regression model fits a set of data, or in other words, the error between the actual and predicted values. The higher the value (or cost), the greater the error (or cost); the lesser the value, the lesser the error. It is represented by the function:

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

Where m is the number of datapoints in the dataset, $h_\theta(x^{(i)})$ is the predicted output of the current model for the ith element of the dataset, and $y^{(i)}$ is the actual value of the ith value of the dataset.
Code Implementation (Python):

[ ]:
```
def computeCost(X,y,theta):
    m = X.shape[0]
    error = X.dot(theta) - y
    error = error**2
```

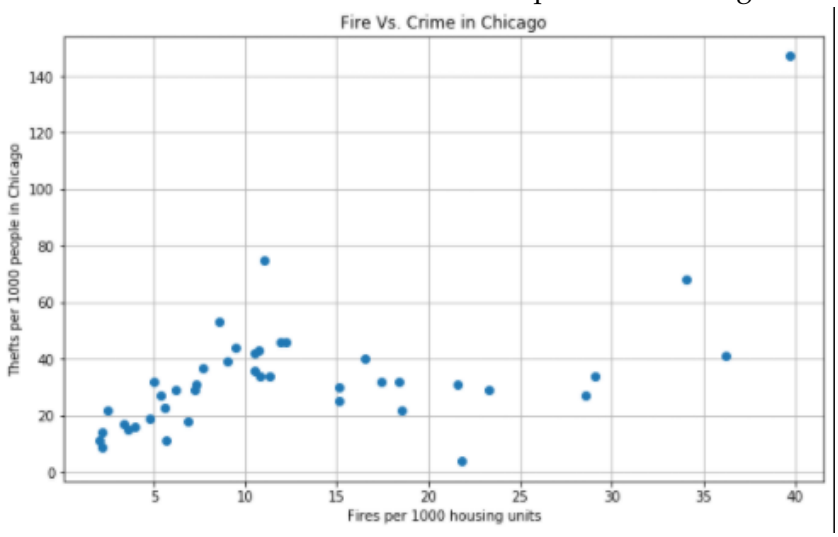
```
    totalsqrError = 0
    for i in error:
        totalsqrError+=i
    J = 1/(2*m)*totalsqrError
    return J
```

Gradient Descent
Gradient Descent is an algorithm used to find the model of best fit for a particular machine learning model. It works by assigning an arbitrary value to theta, the slope and y-intercept in the case of linear regression, and updating itself upon every iteration by subtracting the derivative of the cost function multiplied by the learning rate of the algorithm for a set number of iterations. After the completion of every iteration of gradient descent, the model that it is optimizing becomes less wrong in predicting the given data until the value of theta is at the lowest possible value. The gradient descent algorithm is represented by the following function:

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

In this equation, θ_j represents the parameters of the model. In univariate linear regression, theta is a vector that stores both the slope and y-intercept of the model. α represents the learning rate of gradient descent. α is a value that is arbitrarily chosen by the programmer that determines how many iterations gradient descent will have to go through in order to converge. Smaller values of α will result in more iterations of gradient descent and are generally more accurate but take significantly longer to run than larger step sizes. Lastly, $\frac{\partial}{\partial \theta_j} J(\theta)$ represents the partial derivative of the cost function $J(\theta)$ with respect to theta.

We have decided to show a full implementation of linear regression. To do this, we have selected a dataset from the internet to run our model on. The dataset displays the amouut crime in Chicago as a function of the number of house fires per 1000 housing units. Here is a scatterplot of our data:



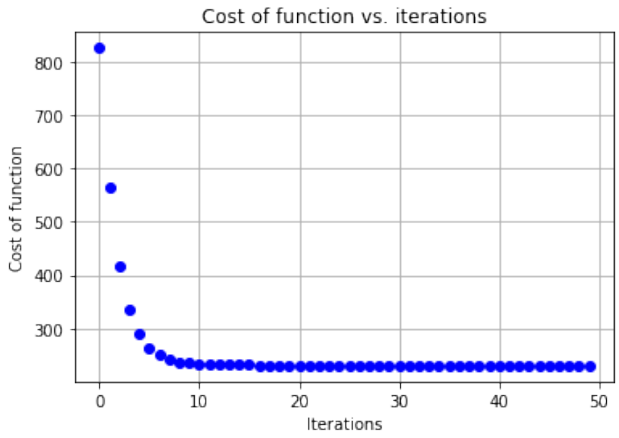
As seen, the data show a clear(although not perfect) linear pattern.
Below, we have implemented gradient descent as a Python function. It takes the parameters X,a matrix containing all of the x values of the matrix, y, a matrix containing all of the y-values of the dataset, alpha, the learning rate which is α from the previous equation, and *iterations*, which is the number of iterations that gradient descent will perform. The function returns, *costProg*

an array that contains the cost from every iteration of gradient descent, *thetaProg*, an array that contains the value of theta at every iteration of gradient descent, *theta*, the final value of theta and X and y which are the feature and target values of the dataset respectively.

[ ]:
```
def gradientDescent(features,targets,alpha,iterations):
#     initialtheta = np.zeros((features.shape[1],1))
    m = targets.size
    features = features.reshape(features.shape[0],1)
    targets = targets.reshape(targets.shape[0],1)
    features = np.insert(features,0,1,axis=1)
    theta = np.zeros((features.shape[1],1))
    costProg = []
    thetaProg = []
    i = 0
    while(i<iterations):
        tempTheta = theta
        costProg.append(computeCost(features,targets,theta))
        for k in range(theta.size):
            cost = (alpha/m)* np.sum((features.dot(theta)-targets)*np.
 ↪array(features[:,k]).reshape(m,1))
            tempTheta[k] = tempTheta[k] - cost
        theta = tempTheta
        thetaProg.append(theta)
        i = i+1
    return costProg,thetaProg,theta,features,features
```

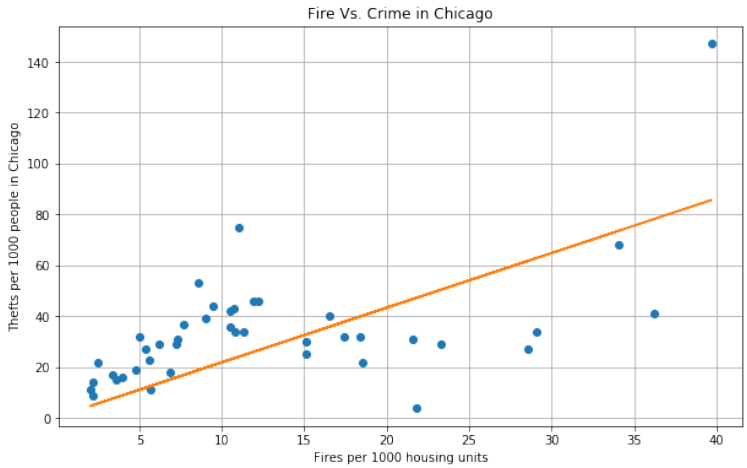
To show the progression of our theta, we plot the cost of it against the number of iterations. As seen in the following visual, the cost of theta at the first iteration was extremely high, which means that our initial model was very wrong. Upon each iteration, as theta became increasingly correct, the cost of theta went down. It is clearly shown that around the tenth iteration, the cost of the model had essentially converged, meaning that our theta values were extremely close to ther optimal values of theta.

[3]:
```
plt.plot(range(len(costProg)),costProg,'bo')
plt.grid(True)
plt.title('Cost of function vs. iterations')
plt.xlabel('Iterations')
plt.ylabel('Cost of function')
plt.show()
```



Now finally, with the optimal value of theta as found by our gradient descent function, we can plot our model against the data.

[4]:
```
plt.figure(figsize = (10,6))
plt.plot(xPlot,finY,'o')
plt.plot(xPlot,finTheta[1]*xPlot+finTheta[0],)
plt.xlabel('Fires per 1000 housing units')
plt.ylabel("Thefts per 1000 people in Chicago")
plt.title("Fire Vs. Crime in Chicago")
plt.grid(True)
plt.show()
```



The model above shows our model when plotted on a graph against the our data. This is the final model that our linear regression algorithm calculated.

3 Proofs

1. **Proof that the Cost Function for Univariate Linear Regression is convex**
   Let the cost function be defined by:

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Definitions
For all $m \in \mathbb{N}$, let A be an arbitrary function where $A = \{(x_1, y_1), (x_2, y_2) \dots (x_m, y_m)\}$.
Note that set A represents the number of points in the given data set.
Let h_θ be the linear function that we are trying to compute the cost of.
$h_\theta(x) = \theta_0 + \theta_1 x$ where θ_0 is the y-intercept and θ_1 is the slope.
Proof: Let $x_i \in \mathbb{R}^n$ be the ith training example, let X be a matrix with its ith row denoted as x_i^T, and let y be a column vector with its ith entry denoted as y_i.
For $J : \mathbb{R}^n \to \mathbb{R}$,

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=0}^{m} (x_i^T \theta - y_i)^2 \\ &= \frac{1}{2m} \| X\theta - y \|_2^2 \end{aligned}$$

For a twice differentiable function, f, with $n \in \mathbb{N}$ inputs, let its Hessian matrix be defined as

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

To prove convexity of f, its Hessian matrix must be positive definite, or in other words, consist of all positive eigenvalues.
Given the function,

$$f(\theta) = \frac{1}{2m} \| \theta \|_2^2$$

its Hessian matrix is positive convex, as

$$\nabla^2 f(\theta) = \frac{1}{m} I$$

with I being the identity matrix.
Thus, we can conclude that the cost function is convex.

2. **Proof for the convergence of gradient descent with fixed step size**
Definitions
For function f : $\mathbb{R}^d \to \mathbb{R}$,
Gradient decent: For an $x_0 \in \mathbb{R}^d$ and step size $t \geq 0$, gradient descent (GD) refers to the following algorithm :

$$x_{i+1} = x_i - t \nabla f(x_i)$$

Convex: f is convex if for all x, y $\in \mathbb{R}^d$ and $0 \leq \lambda \leq 1$,

$$f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

L-Lipschitz: f is L-Lipschitz if for all x, y $\in \mathbb{R}^d$

$$\| \nabla f(x) - \nabla f(y) \|_2 \leq \| x - y \|_2$$

Let function f be Lipschitz convex, the following properties hold true:
Property 1: For all x, y $\in \mathbb{R}^d$,

$$f(x) + \langle \nabla f(x), y - x \rangle \leq f(y)$$

Property 2: For all x, y $\in \mathbb{R}^d$,

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|_2^2$$

Property 3: For $y_1, y_2 ... y_k \in \mathbb{R}^d$,

$$f(\frac{y_1 + y_2 ... + y_k}{k}) \leq \frac{f(y_1) + f(y_2) ... + f(y_k)}{k}$$

Now prove that gradient descent converges to a global optimum for Lipschitz convex functions, i.e
For a function f that is L-Lipschitz and convex, let x^* be the x at which f(x) is at its minimum, or arg $\min_x$f(x). Then for GD with the step size $t \leq 1/L$:

$$f(x_k) \leq f(x^*) + \frac{\|x_0 - x^*\|_2^2}{2tk}$$

Proof:
Take the equation of GD, $x_{i+1} = x_i - t\nabla f(x_i)$, and apply Property 2:

$$\begin{aligned}
f(x_{i+1}) &\leq f(x_i) + \langle \nabla f(x_i), x_{i+1} - x_i \rangle + \frac{L}{2} \|x_{i+1} - x_i\|_2^2 \\
&= f(x_i) + \langle \nabla f(x_i), x_i - t\nabla f(x_i) - x_i \rangle + \frac{L}{2} \|x_i - t\nabla f(x_i) - x_i\|_2^2 \\
&= f(x) + \langle \nabla f(x), -t\nabla f(x) \rangle + \frac{L}{2} \|-t\nabla f(x_i)\|_2^2 \\
&= f(x) - t \|\nabla f(x_i)\|_2^2 + \frac{Lt^2}{2} \|\nabla f(x_i)\|_2^2 \\
&= f(x_i) + t(Lt/2 - 1) \|\nabla f(x_i)\|_2^2 \\
&\leq f(x_i) + t(L(1/L)/2 - 1) \|\nabla f(x_i)\|_2^2 \\
&= f(x_i) - \frac{t}{2} \|\nabla f(x_i)\|_2^2
\end{aligned} \tag{1}$$

Next, we can bound $f(x_i)$ in terms of $f(x^*)$, the optimal objective value. By Property 1:

$$\begin{aligned}
f(x_i) + \langle \nabla f(x_i), x^* - x_i \rangle &\leq f(x^*) \\
f(x_i) - \langle \nabla f(x_i), x_i - x^* \rangle &\leq f(x^*) \\
f(x_i) &\leq f(x^*) + \langle \nabla f(x_i), x_i - x^* \rangle
\end{aligned} \tag{2}$$

Since we know that $x_{i+1} = x_i - t\nabla f(x_i)$, we can easily deduce that $\nabla f(x_i) = \frac{1}{t}(x_i - x_{i+1})$.

Using this, and the previous two deductions:

$$
\begin{aligned}
f(x_{i+1}) &\leq f(x_i) - \frac{t}{2}\left\|\nabla f(x_i)\right\|_2^2 \\
&\leq f(x^*) + \langle \nabla f(x_i), x_i - x^* \rangle - \frac{t}{2}\left\|\nabla f(x_i)\right\|_2^2 \\
&= f(x^*) + \langle \frac{1}{t}(x_i - x_{i+1}), x_i - x^* \rangle - \frac{t^2}{2t}\left\|\nabla f(x_i)\right\|_2^2 \\
&= f(x^*) + \frac{1}{t}\langle x_i, x_i - x^* \rangle - \frac{1}{t}\langle x_{i+1}, x_i - x^* \rangle - \frac{1}{2t}\left\|t\nabla f(x_i)\right\|_2^2 \\
&= f(x^*) + \frac{1}{t}\langle x_i, x_i - x^* \rangle - \frac{1}{t}\langle x_i - tf(x_i), x_i - x^* \rangle - \frac{1}{2t}\left\|t\nabla f(x_i)\right\|_2^2 \\
&= f(x^*) + \frac{1}{t}\langle tf(x_i), x_i - x^* \rangle - \frac{1}{2t}\left\|t\nabla f(x_i)\right\|_2^2 \\
&= f(x^*) - \frac{1}{2t}\left(-2\langle tf(x_i), x_i - x^* \rangle + \left\|t\nabla f(x_i)\right\|_2^2\right) \\
&= f\left(x^*\right) + \frac{1}{2t}\left\|x_i - x^*\right\|_2^2 - \frac{1}{2t}\left(\left\|x_i - x^*\right\|_2^2 - 2\left\langle t\nabla f\left(x_i\right), x_i - x^* \right\rangle + \left\|t\nabla f\left(x_i\right)\right\|_2^2\right) \\
&= f\left(x^*\right) + \frac{1}{2t}\left\|x_i - x^*\right\|_2^2 - \frac{1}{2t}\left(\left\|x_i - x^*\right\|_2^2 - 2\left\langle t\nabla f\left(x_i\right), x_i - x^* \right\rangle + \left\|t\nabla f\left(x_i\right)\right\|_2^2\right) \\
&= f\left(x^*\right) + \frac{1}{2t}\left\|x_i - x^*\right\|_2^2 - \frac{1}{2t}\left\|x_i - x^* - t\nabla f\left(x_i\right)\right\|_2^2 \\
&= f\left(x^*\right) + \frac{1}{2t}\left(\left\|x_i - x^*\right\|_2^2 - \left\|x_{i+1} - x^*\right\|_2^2\right)
\end{aligned}
\tag{3}
$$

This equation holds on every iteration of gradient descent, so if we take the sum over all iterations, we get:

$$
\begin{aligned}
\sum_{i=0}^{k-1} f(x_{i+1}) - f(x^*) &\leq \sum_{i=0}^{k-1} \frac{1}{2t}(\left\|x_i - x^*\right\|_2^2 - \left\|x_{i+1} - x^*\right\|_2^2) && (4) \\
&= \frac{1}{2t}(\left\|x_0 - x^*\right\|_2^2 - \left\|x_k - x^*\right\|_2^2) && (5) \\
&\leq \frac{\left\|x_0 - x^*\right\|_2^2}{2t} && (6)
\end{aligned}
$$

Since we know that f decreases on every iteration:

$$
\begin{aligned}
f(x_k) - f(x^*) &\leq \frac{1}{k}\sum_{i=0}^{k-1} f(x_{i+1}) - f(x^*) && (7) \\
&\leq \frac{\left\|x_0 - x^*\right\|_2^2}{2tk} && (8)
\end{aligned}
$$

It follows that:

$$
f(x_k) \leq f(x^*) + \frac{\left\|x_0 - x^*\right\|_2^2}{2tk}
$$

Therefore, we have proven that gradient descent converges to a global optimum for Lipschitz convex functions where the step size $t \leq 1/L$.

4 Conclusion

In conclusion, the application of mathematics in machine learning are essential in understanding the conceptual basics of fundamental algorithms. Through this experience, we further expanded our knowledge on machine learning algorithms and their implementations and subsequent applications, while practicing proof writing. Exploring the process by which these equations were derived helped us better grasp the reasons of how they work the way they do.

5 References

References

[1] K. Hao. What is machine learning? https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/, Nov. 2019. Accessed: 2020-12-10.

[2] R. Meka. Notes on convergence of gradient descent. https://raghumeka.github.io/CS289ML/gdnotes.pdf. Accessed: 2020-12-10.

[3] L. Miller. Machine learning week 1: Cost function, gradient descent and univariate linear regression. https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5f Jan. 2018. Accessed: 2020-12-10.

[4] A. Ng. Machine learning. https://www.coursera.org/learn/machine-learning/supplement/2GnUg/gradient-descent, Aug. 2011. Accessed: 2020-12-10.

[5] P. Pandey. Understanding the mathematics behind gradient descent. https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e, Mar. 2019. Accessed: 2020-12-10.

[6] T. Ray. Why gradient descent works for linear regression. https://towardsdatascience.com/understanding-convexity-why-gradient-descent-works-for-linear-regression-aaf763308708. Accessed: 2020-12-10.

[7] S. Saxena. Mathematics behind machine learning – the core concepts you need to know. https://www.analyticsvidhya.com/blog/2019/10/mathematics-behind-machine-learning/, Oct. 2019. Accessed: 2020-12-10.

[8] R. Tibshirani. Lecture 6: September 12. https://www.stat.cmu.edu/~ryantibs/convexopt-F13/scribes/lec6.pdf, 2013. Accessed: 2020-12-10.

[9] S. Yagcioglu. Classical examples of supervised vs. unsupervised learning in machine learning. https://www.springboard.com/blog/lp-machine-learning-unsupervised-learning-supervised-learning/, May 2020. Accessed: 2020-12-10.