Machine Learning

**Lecture 0: Why Machine Learning?**

* ML is a type of AI that allows software applications to become more accurate by predicting future outcomes using some dataset
* Use case: reccomnedation engines, fraud dectection, spam filtering, malware threat detection etc
* Why is it important?
  + Gives idea of user trends in customer behaviour and business operation patterns
  + Supports development of new products
* 4 basic approaches:
  + Supervised learning
  + Unsupervised learning
  + Semi-supervised learning
  + Reinforment learning
* Supervised learning
  + Data scientists supply algorithms with labeled training data
  + Input and output of algorithm is specified
  + Ex. regression modeling, binary classification, ensembling
* Unsupervised learning
  + Do not require data to be labeled
  + Machine sifts through unlabeled data to look for patterns
  + Ex. deep learning, clustering
* Reinforcment learning

**Lecture 1: Linear Regression Single Variable**

* Linear regression model uses equation: y=mx+b
* Python code:
  + Create data frame using csv file *(df = pd.read\_csv(‘test\_csv.csv’))*
  + Import linear model from sklearn *(from sklearn import linear\_model)*
  + Create linear model (reg = linear\_model.LinearRegression())
  + Create line of best fit (reg.fit(df[[‘area’]], df[.price]))
  + Predict future values (reg.predict(3300))

**Lecture 2: Linear Regression Multiple Variables**

* 1 dependent variable is determined by many (2 or more) independent variables
* Ex.
  + Price = m1 \* area + m2 \* bedrooms + m3 \* age + b
* Independent variables are also called features
* m1, m2, m3 are also called coefficients
* Data preprocessing: Handling NA values
  + Figure out what you want to change NA values too
  + Ex. If one of the bedroom number values is NA, one approach is to find the median of all the bedroom values and sub it into all the NA values
  + In python use: (df.column.fillna(0))
* Create line of best fit
  + Use sklearn function *reg.fit*
  + reg.fit => (independent variables, dependent variables)
  + Python code: *reg.fit(df[[‘rooms, age, area’]], df.price)*
* Look at coeffiencents and intercept
  + Python code: *reg.coef\_ , reg.intercept\_*
* Predict futre values
  + Python code: *reg.predict[[2000, 3, 2]]*

**Lecture 3: Gradient Descent and Cost function**

* Gradient Descent is an algorithm that finds the best line fit for given training data set
* **A picture containing text, clock, gauge

  Description automatically generated**MSE (cost function) is the difference between the actual output value – predicted output value, squared divided by n
* Learning Rate: Is the adjustment factor of how much each paramter will move by to eventually get to the optimal value
* SSR (loss function)
  1. **Text

     Description automatically generated**Summation of the (difference between the actual output value – predicted output value) squared
* Gradient Descent steps (one variable – specifically the y-intercept for linear relation):
  1. Find a metric to evaulte how well a line fits the data (Loss function)
     + In this example, let’s use Sum of Squared residuals as Loss function
  2. Take Derivative of the loss function with respect to that one variable
     + In this example, partial derivative of y-intercept
  3. Pick a random value for the intercept
     + In this example, you can start at y-intercept = 0
  4. Calculate derivative with that chosen random value (fancy word for slope)
  5. Plug that slope into step size calculation
     + Step size = slope \* learning rate
  6. Calculate new intercept
     + New intercept = old intercept – step size
  7. Plug new intercept into derivative an repeat everything until step size is close to zero
* Note: Gradient descent is very sensitve to the learning rate
  1. Certain learning rates will not find the minimum loss function
* Note: Sum of the squared residuals is just one type of loss function
  1. Gradient descent works the same way, with any type of loss function
* Gradient Descent (General method for multiple parameters)
  1. Take the derivative of the loss function for each parameter in it
     + This means finding the partial derivative with respect to each feature
     + ML lingo: ‘*Take the gradient of the loss function*’
  2. Pick random values for the parameters
  3. Plug the parameter values into the derivatives
  4. Calculate the step sizes:
     + Step size = slope \* learning rate
  5. Calculate the new parameters:
     + New Parameter = old parameter – step size
  6. Repeat step 3 until step size is very small or you reach maximum number of steps
* Problem: when there are millions of data points, algorithm can take a long time
* Solution: Use Stochastic Gradient Descent
  1. Uses a randomly selected subset of the data at every step rather than full dataset
  2. This reduces the time to calculate the derivatives of the loss function

**Lecture 4: Save and load model using Joblib and Pickle**

* Pickle module allows us to serialize python object into a file
* Python code:
  + *import pickle*
  + put model into a file: *with open(‘model\_pickel’,’wb’) as f: pickle.dump(model,f)*
  + retrieve model: *with open(‘model\_pickel’,’rb’) as f: mp = pickle.load(f)*
* You can also use Joblib which is more efficient on objects that have large numpy arrays
* Python code:
  + *import joblib*
  + put model into file: *joblib.dump(model, ‘model\_joblib’)*
  + Load model: mj = *joblib.load(‘model\_joblib’)*
* **Lecture 4: Save and load model using Joblib and Pickle**