Machine Learning Foundation

Created	@Jan 18, 2021 1:41 PM
Created By	K Khanh Vương
Last Edited By	K Khanh Vương
Last Edited Time	@Feb 8, 2021 5:51 PM
Module	Introduction to Machine Learning
Status	
Type	Introduction to Machine Learning

Linear Regression Problem

Definition of Linear Regression

- **Linear Regression** has a wide-range use cases. Think of it when we want to compute the `Linear` correlation between variables.

Equation form of Linear Regression

- **Regression** can take the form of any linear equations, with $W(w1, w2, ...)$ as the parameters, sometimes we call it `Weight`. Our mission is to find the best fitting W, so that we can approximate the best result.
  - **x-axis**: feature, covariate, predictor, or Independence variable
  - **y-axis**: observation, respond, dependence variable of **x-axis**

Evaluate Linear Regression model

- To know how good a **Linear Model** is, compute:
  - **Residual Sum of Squares (`RSS`)**
  - **Root Mean Square Error (`RMSE`)**

Overfitting Problem

- Sometimes, we may face off with **Overfitting** problem. To avoid this, we usually split the data into two parts:
  - **Training Set**
  - **Test Set**
  - This will allow us to control how fit the model is base on `RSS` of the prediction on `Test Set`
- We will want to minimize the weight $W(w_1, w_2, ...)$ on the `Training Error` and `Test Error`, this will lead to minimize the `Loss Function`

The flexibility

- The higher dimension of parameter $W(w_1, w_2, ...)$, the more flexible the model is. Think of the derivate of the model, each parameter in $W(w_1, w_2, ...)$ will represent an inflection point (an extreme). The more extreme points, the more the Linear Line becomes fit with the original data. This may be good at some point, but it gets worse over time because it fits too well for a particular data set and loses the ability to predict the future.

- `Training Error` will be decreased, the model will get better and better fit the training data
- `Test Error` will be decreased, but will be increased at some points

Reference

- **Notebook:** https://colab.research.google.com/drive/1FcZugLyQK1k3T5XCU1_KZ_F996zuc4pc?usp=sharing

Classification Problem

Linear Classifier

- `Linear Classifier` is a problem that find the best parameter $W(w_1, w_2, ...)$ for each word in the sentence

$$Score = w_1 x_1 + w_2 x_2 + ...$$

- Have a threshold that separate outputs, divide the results in many groups

Decision Boundaries

- **Decision Boundaries** is a line, a plane, or a hyperplane in the **Scatter Plot** between variables. It is the equation with specific parameter $W(w_1, w_2, ...)$ that separate results into groups

Evaluate Classification Model

- Different from `Linear Regression` that use `RMSE` to calculate the prediction errors, here, we calculate the `True/False` probability of the predicted results on the `Test Set`

$$error = \frac{\#\text{mistake}}{\#\text{sentiments}}$$

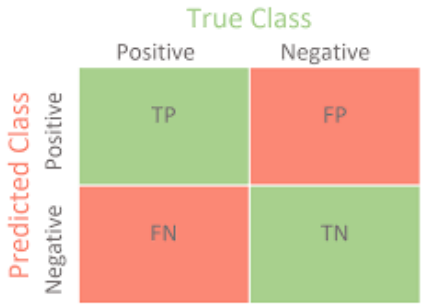
$$accuracy = \frac{\#\text{correct}}{\#\text{sentiments}}$$

- To know how good our model is:
  - Compare the $accuracy$ with the probability of random guess

    - Check for `Class Imbalance`, because if there's a majority class, then it will highly affect our $accuracy$

- Whether we are working on a `Logistic Regression` problem, we can evaluate the model by the mean of `ROC Curve`

Confusion Matrices



- The $accuracy$ will be the sum of all `True Positive` and `True Negative` in the `Confusing Matric`

The amount of input data problem

- **Bias** is the range between the `Test Error` and 0% in the `(Test Error, Amount of data)` graph
- The less the `Bias`, the less the `Test Error`
    - Why `Test Error` ? Because `Training Error` is used for training progress, and we can only compute the accuracy on the test progress.
- The curve of two graph: one with a little parameters and one with more parameters will cross somewhere. This is the place when the model with more parameters perform better than the other. Note that, data must be good, or else we will get worse result.

Class Probabilities

- After our model returned predicted result, we may want to know how consistency the result is. Think of `Class Probability`:

$$P(label|input)$$

- Predicting probabilities or level of confidence is extremely important. For example when we know the probability, we can make decisions like, what is a good decision boundary that trades off false positives and false negatives, and balance between the two

Reference

- **Notebook**: https://colab.research.google.com/drive/1szFx1mTj1zbfQz16hB0V0fSbgz84j6XO?usp=sharing

Clustering Problem

Clustering Definition

- `Clustering` is the problem that group many observations that related to each other into clusters
- There are two main tasks:
  - Compute the similarities between observations
  - Group those related observations base on there're similarities

Represent the data in Sentiment Analysis

- `Bag of Words` is the idea that compare the similarities between words count vectors
- However, there is a problem with words count vector. The longer the document, the higher the word count vector, so it will affect our result, and the bias is more likely toward the longer document. To avoid this, be sure to normalize every vector before compute the word count vector
- To normalize a vector, simply take each element of the vector and divide it by the **Norm** *(the modulus of the vector)*

Measuring the similarities between words count vectors

- To compute the similarities, simply take the dot product of two vectors

$$\sum \frac{x_i}{|x|} \frac{y_i}{|y|}$$

Emphasize the important words

- We always want to emphasize those important words, usually are those words that dominated by common words.
- Moreover, important words are those words that appear infrequently in the corpus. So, we will want to discount the weight of the words every time that word appear in any docs of the corpus *(Inverse Document Frequency)*
- To calculate the important of every words in a document, compute `TF-IdF` **(Term Frequency - Inverse document Frequency)** vector:
  - First, compute it's `TF`. `TF` is just the word count vector of that document
  - Then down weight this vector base on **IDF**

$$IdF = \lg \frac{\#\text{doc}}{1+\#\text{DocsUsingWord}}$$

K - Nearest Neighbor

- The main idea is to compute the distance metrics between observations, then return the top **K** smallest distance values from the query input (meaning those observations that have the most similarities with the input query observation)

Cluster Definition

- A **Cluster** will be defined by it's *center* and it's *shape/spread*. **Cluster** will allow us to produce faster recommend instead of looking at every documents in the corpus
- To assign an observation to a `Cluster`, compute the score or the distance from that observation to the center or the edge of each `Cluster`, then return the `Cluster` with smallest distance

K-means Algorithm

- The concept of this algorithm is that, we want to split the data into **K Clusters,** then we will compute the **mean** of the `Clusters` to finally find the center
  - First, we will assume that we have `K Clusters`, and put `K` random center
  - Assign any close observations to each random center we just created
  - Use `Voronoi Tessellation`, a way to represent `Clusters Region`

- Recalculate the center for each `Cluster` by taking the **mean** of all observations that assigned to that `Cluster`
- Iterate the final step again and again until `convergence`