AIML - CS 337

Lecture 3: Linear Algebra Review, Classification Task

11 August 2022

Lecturer: Abir De *Scribe: Advait, Ganesh, Suman, Vaibhav*

This lecture starts with a review of Linear Algebra and some of its uses in Machine Learning. Later on, the general Machine Learning problem of Classification is discussed.

1 Linear Algebra (continued)

Apart from the general usage of Linear Algebra in Machine Learning as a tool to represent features (as vectors/matrices) and weights (as matrices), there exist many tasks requiring deeper insights.

1.1 Time Dependence in Networks

Consider a classification task where we get sequences as input. In a normal classification task, features are of the form **(*feature*, *label*)** while for a sequence, there is a time component and each interval has a feature vector - $x_t, x_{t+1}, \ldots$ with possible dependence across features. One possible relationship is a linear dependence of the features at time $t+1$ on the features at time t.

Suppose we have a network where edges represent influence of nodes over each other. This influence travels over time and the information/data at one node affects its neighbors and other nodes in the future. Social networks (on Facebook/Instagram) can be represented in this form and the information in the nodes can possibly be sentiment of people towards politicians or extent of spread of misinformation.

Consider a graph similar to the one shown to the right. Suppose that nodes i and j share an influence over each other with strength w_{ij} (not necessarily same as w_{ji}). For instance, the influence of 1 over 6 is twice that of 2 over 6 in the example graph (which is undirected, hence symmetric). These influences can be represented in a matrix $\mathbf{W}$ of size $N \times N$ where N is the number of nodes in the graph. The elements on the diagonal w_{ii} will represent remembrance of the previous state at some node.

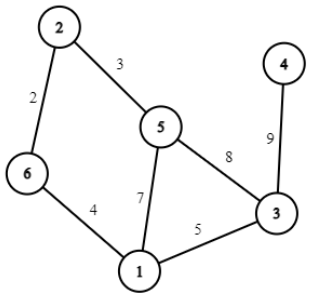


Figure 1: Example Graph

Suppose the state at some time t of the i^{th} node is $\mathbf{x_t^i}$. These state vectors can be stacked row-wise in a matrix $\mathbf{X_t}$ which evolves with time. Assume that the state changes as following -

$$\mathbf{x_{t+1}^i} = \sum_{j=1}^{N} w_{ij} \mathbf{x_t^j}$$

The evolution of states can then be represented as - $\mathbf{X_{t+1}} = \mathbf{W X_t}$

1.2 Stochastic Matrices

Under some special conditions, the state denoted by $\mathbf{X_t}$ will converge. Such is the case of stochastic matrices.

Definition 1.1. Matrix $\mathbf{W}$ is **stochastic** if all entries are non-negative and sum of all entries in a column equals 1. $\mathbf{W}$ is further **positive** if all entries are positive numbers.

Lemma 1.2. *Properties of eigenvalues of a stochastic matrix* $\mathbf{W}$ *are given below -*

1. 1 *is an eigenvalue of* $\mathbf{W}$.
2. *If* λ *is an eigenvalue (real/complex) of* $\mathbf{W}$, $|\lambda| \leq 1$.

Some more important results related to stochastic matrices -

Definition 1.3. For a stochastic matrix $\mathbf{W}$, a *steady state* is an eigenvector with eigenvalue 1 whose entries are positive and add to 1.

Theorem 1.4 (Perron-Frobenius Theorem)**.** *For a positive stochastic matrix* $\mathbf{W}$*, there exists a unique steady state vector* v *which spans the* 1*-eigenspace. Also, for any vector* v_0 *with entries adding to* s*, iterates* $v_1 = Av_0, v_2 = Av_1, \ldots, v_{t+1} = Av_t, \ldots$ *tend to* sv *as* t *gets large.*

More discussion can be found at [1].

1.3 Average Consensus

In this case, the weight matrix is such that all neighbors get equal weights and non-neighbors don't have any contribution. Suppose the graph is G and its edge set is E.

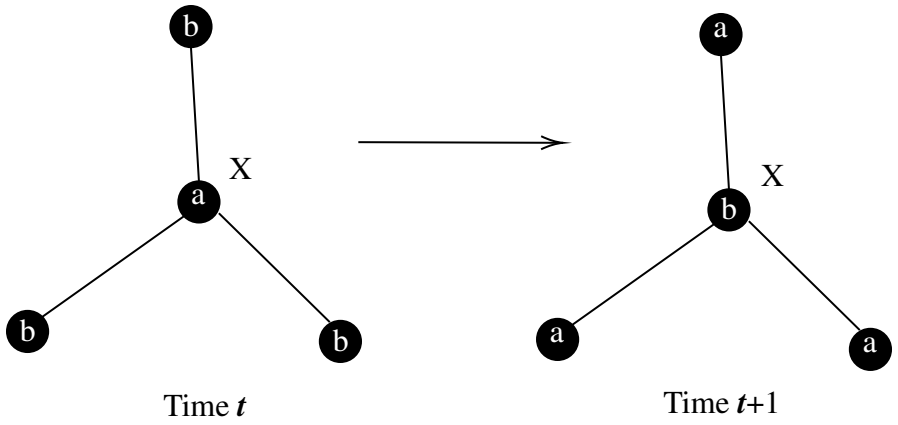
$$\mathbf{x_{t+1}^i} = \frac{\sum_{j \in N(i)} \mathbf{x_t^j}}{|N(i)|}$$

Upon converting the equation to matrix form, we get the weight matrix $\mathbf{W}$ such that $w_{ij} = \frac{1}{d_i}$ (degree of vertex i) if $(i, j) \in E(G)$, else $w_{ij} = 0$. Here, each sum rows to 1 and this is a right stochastic matrix.

In this case, $\mathbf{1}$ is an eigenvector of $\mathbf{W}$ with eigenvalue 1 because this multiplication just amounts to adding up every row which we already have as 1.

1.4 Non-convergence of state

We can create instances of graphs where convergence is not reached. Consider the below graph with a central node $\mathbf{X}$ connected to some nodes (a star type network) . Suppose $\mathbf{X}$ has a state of $\mathbf{a}$ while all its neighbors have a value of $\mathbf{b}$. If the node state relation is a simple average over neighbors (average consensus), these values will keep oscillating - $\mathbf{a}$ on $\mathbf{X}$, $\mathbf{b}$ on neighbors to $\mathbf{a}$ on neighbors, $\mathbf{b}$ on $\mathbf{X}$. Convergence will never be reached in this case.



2 Introduction to the Classification Task

Machine Learning deals with multiple tasks which can be broadly classified into Classification, Regression, Clustering etc. Of these, classification is a very important task & is specified below -

Task 2.1. *Given K items, as features of the form $X_1, X_2, \ldots, X_K$, classify them into one of N possible labels $1, 2, \ldots, N$.*

Solution Approach - To solve the above task, we will need a function $\mathbf{H}$ such that $\mathbf{H}(X_i) = Y_i$ where Y_i is the expected label for the item with features X_i. The labels are defined by users, i.e., we don't know the definition or the meaning of a label. Suppose that no information about the relation between labels and features is provided to the function. A simple permutation of labels on the user side will lead to mistakes in the labels resulting from the function $\mathbf{H}$ even though the input didn't change. Hence, some information about the relation between labels and features is required. The easiest way to get this information is through examples, which is the first step in the solution method.

1. Ask the user for information about labels in the form of examples i.e. $\{X_i, y_i\}_K$, formally called the **Training Set**.

The user will provide us some already classified data (examples) and once we have the example pairs, the next step is to learn from them to be able to predict the labels for **unseen examples**.

2. Create an algorithm **ALG** which *learns* from provided examples and is able to do *inference* i.e. predict labels for arbitrary input.

Once the algorithm **ALG** is ready, we need to verify if it indeed works well for unseen inputs. For this, we would need some more examples as pairs to test results.

3. Verify **ALG** on examples not included in the Training Set (checking the correctness of **ALG**). This set of examples is formally called the **Validation Set**. We don't use this set to edit the algorithm, it just helps accept/discard an algorithm.

Asking the user for more examples at this stage can be cumbersome because label collection can be expensive with respect to time and money. Hence, all examples are asked for at the start of the process and the full dataset is divided into the 2 portions - **Training Set** & **Validation Set**. Our goal during training is *generalizability* - the algorithm should be as suitable for the Validation set as for the Training set.

Note - We blindly trust the data provided to us by the user i.e., we assume that the data is correct and user doesn't do any mistakes.

References

[1] Dan Margalit and Joseph Rabinoff. *Interactive Linear Algebra*. 2019.