# Problem-Solving With Machine Learning

**Project Part One: Frame a Machine Learning Problem**

**Instructions:** Think of a problem that you want to solve with machine learning. Frame this problem like a data scientist by answering the following questions. Please limit your answers to 100 words or less.

|  |
| --- |
| What is the **context** of the problem? Why is it important to solve this problem? |
| An opinion prediction application related to customer sentiments about autonomous vehicles is suggested [1]. This application will predict how public consider autonomous vehicles i.e., it could be positive, negative, or neutral. In the early stages of new technologies such as autonomous vehicles, there is always some ambiguity as to their potential effectiveness. Organizations may reduce this ambiguity by assessing the customers’ sentiments when consumers ultimately decide to embrace or reject modern technology. Industry practitioners might find this application useful in making investment decisions towards autonomous vehicles technology. |
| What inputs (**features**) would you include? Describe **data types** and possible feature **transformations**. |
| Input=X= Text data taken from tweeter i.e., maximum length of 280 words.  This data is unstructured and will be vectorized using BOW(Bag Of Word) model. Suppose we have 10,000 samples then they will become our sample size. Our features will be number of unique words in our corpus. Let’s say we have 5,000 words then shape of our input is;  (10000,5000) |
| What are the potential outputs (**labels**)? |
| Output=y= Customers’ sentiment i.e., positive, negative, or neutral.  This output will be labelled or one-hot encoded. |
| Are **observations dependent** anyhow? Often, we have either temporal dependency or multiple observations per patient or device or event. This is important for train/test splitting. |
| In BOW model, words are not dependent on each other. It provides arbitrary ordering of words in form of vectors. A temporal dependency might be possible if we consider time for which data is collected. Since, my problem is simple, and it only predicts whether a tweet(text) is positive, negative or neutral therefore; there is no temporal dependency. |
| Is this a **regression**, or **classification** (**binary** or **multiclass**)? Explain. |
| It is multi classification problem since I have three possible outcomes. On dashboard, I will enter a text and as of output, we can get positive, negative, or neutral sentiment response. |
| How might you split the observations into **train** and **test** sets? Are there potential **biases** to look out for? |
| I will use 80% of data as train set.  10% as validation set  10% as test set.  First, I ll train on training set and validate my results using validation (10% data). If I get less accuracy then I will do further preprocessing and model tweaking to improve results. Finally, model will put to test. At this stage, train+validation data is trained on a model and that model will be test on test.  There could be some biasness.   1. With passage of time, there might be new words added to tweeter corpus which my training data does not contain. Due to this, I might face generalization issue. This is also called over-fitting. 2. In my data, there might be class mismatch. I might have high number of neutral tweets about AV. This will make my model to learn patterns related to neutral sentiments better than other two classes. |
| What type of **loss** function might be appropriate for quantifying the error of your algorithm? |
| Multi class cross entropy loss function is used for this problem[2]. This will calculate a score that summarizes the average difference between the actual and predicted probability distributions for all classes (positive, negative and neutral sentiment) [1]. The lower the score, better our model would perform. Generally, a perfect cross-entropy value is 0. |

# Project Part Two: Application and Limitations of k-NN

**Instructions:** Answer the following questions about the k-Nearest Neighbors algorithm. Please limit your answers to fewer than 100 words.

|  |
| --- |
| You have implemented k-NN to learn a decision boundary from a training data set that you know to be noisy. To address the issue of noise, you increase this number of nearest neighbors to nine, which has seemingly reduced the impact of the noise on your decision boundary. However, you notice that significant groups of data points in your training data set are now misclassified by this new decision boundary. **Why** might this happen? **How** can you adjust your algorithm **to** **improve** its **accuracy**? |
| When we have smaller K value, we have a restrained region for a given prediction. In such case, model provides the most flexible fit, which will have low bias but high variance. Our decision boundary will be more jagged [3]. When we increase K=9 then it averages more voters in each prediction and hence is more resilient to outliers. K=9 will have smoother decision boundaries which means lower variance but increased bias [4].  To find most optimal accuracy score, we can run k-NN for various values of K (from 1 to 9). We want a case where we have a balance of variance and biasness. |
| When implementing k-NN, you must choose an appropriate distance function. **What** is the role of the distance function and **why** is it important to the accuracy of your machine learning algorithm? |
| To classify an unknown instance represented by some feature vectors as a point in the feature space, the k-NN classifier calculates the distances between the point and points in the training data set. Usually, the Euclidean distance is used as the distance metric. It helps us to get the closest train data points for which classes are known [5]. |
| **Why** does increasing the number of observations also increase the computing **power** and **time** necessary to run k-NN? **What** possible solution is available to help improve the efficiency of k-NN in such a scenario? |
| KNN can be computationally expensive both in terms of time and storage, if the data is very large because KNN has to store the training data to work. This is generally not the case with other supervised learning models. Also, KNN can be very sensitive to the scale of data as it relies on computing the distances. For features with a higher scale, the calculated distances can be very high and might produce poor results [3,4]. |
| **Describe** the curse of dimensionality. **Why** does k-NN break down in high-dimensional space? |
| As per lectures, In k-NN as the dimensionality of the problem grows (from 1D,2D,3D and so on), the higher-dimensional space is less densely occupied by the training data. we need to search a large volume of space to find neighbors of the test point. The pair-wise distance between points grows as we add additional dimensions [6]. The neighbors may be so far away that they do not actually have much in common with the test point. We can solve this by feature selection or by transforming the data into a lower-dimensional space. |

**References**

1. Kohl, C., Knigge, M., Baader, G., Böhm, M., & Krcmar, H. (2018). Anticipating acceptance of emerging technologies using twitter: the case of self-driving cars. *Journal of Business Economics*, *88*(5), 617-642.
2. Loss Function: <https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>
3. For KNN: <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>
4. For KNN k-reduction: <https://towardsdatascience.com/k-nearest-neighbors-94395f445221>
5. Distance Function in k-NN: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4978658/>
6. Curse of dimensionality: <https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-7d64634015d9>