KNNs & Naive Bayes

Charlie Wu*

November 2019

1 Introduction to KNNs

The K-Nearest Neighbors (KNN) algorithm is a *simple classifier and regressor* that does not make any preliminary and inherent assumptions about data. In short, a KNN finds the k-nearest neighbors to a specific feature vector using a distance metric, and uses the most common of these k neighbors to classify the specific unlabelled vector. A KNN takes in labelled training data and tries to adapt to the composition of the data in order to classify novel data. This is useful as many new real-world data need to be judged based off of similarity to existing clusters of data.

2 KNN Classification

The KNN algorithm used for classification is structured as follows:

1. Given a set of training data with features $x_1, x_2, ..., x_n$ and labels y, one unlabelled feature vector to be classified, and a manually defined value k.
2. The "training" phase of this algorithm consists of only storing these feature vectors and labels, and initializing a similarity function to judge the similarity (closeness, distance) of two given vectors.
3. Classification is done through an $O(N)$ search of all the given training feature vectors, sorting the results by similarity (taken from the similarity metric).
4. The first k values returned (the k most similar feature vectors in the training data) are the "k-nearest neighbors".

Put simply, the unlabelled data can take on the most common class of the k-nearest neighbors. This system of choosing the mode is known as "majority voting".

*Naive Bayes based on Sylesh Suresh's lecture of the same name

In this example, all of the points are plotted on the 2d plane, so each feature vector contains two continuous features (plotted as x and y values). The green dot is being classified, and the red and blue dots signify different classes that the points take on. If $k = 3$, the three most similar (closest) points are the three points bounded in the solid black line. In this case, the KNN would classify the green point as being of the red class.

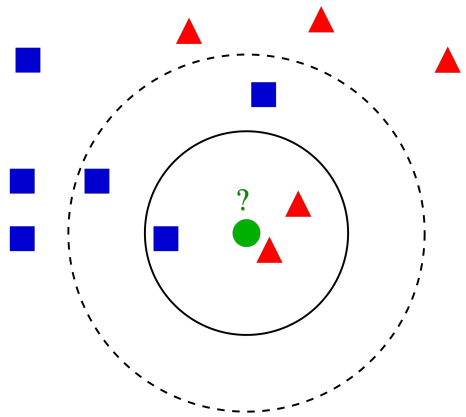

Figure 1: KNN Classification example

3 Similarity/Distance Metric

One key fundamental of a KNN algorithm, as you may have noticed, is the ability to judge the similarity of two feature vectors. This is most commonly done with the Euclidean distance (think 'distance formula' in geometry, applied to every feature in the feature vectors). The Euclidean distance works well when applied to datasets with features involving continuous variables, but fails when the features are discrete. In this case, another metric may be applied. For instance, the Hamming distance, which measures the overlap of two discrete feature vectors. With Hamming, the distance between the words "Amazin" and "Amazon", would be 1 (the only difference being the i and the o), while the distance between "games" and "named" is 2.

4 Weighted Voting

In a training dataset that may contain an skewed class distribution, weights can be applied to the "majority voting" system of classifying the unlabelled data. This new method can apply a weight inversely proportional to the similarity metric calculated to each of the k-nearest neighbors, better accounting for the unequal class variation.

5 KNN Applications

KNNs can be most commonly found in various recommendation systems. Due to their access to large amounts of consumer data, large companies often are able to train KNNs to make the best recommendations of similar or well-liked products to consumers, given enough history of that consumers' interests. Some examples can include Youtube recommending different channels to watch based on a person's watch history and current subscriptions. More specific recommendations can also be made on a large scale, such as in the case of Spotify creating a daily mix of new songs that might appeal to a user.

6 Introduction to Naive Bayes

Naive Bayes is a *simple supervised classifier* based on Bayes' theorem that, despite its assumption that there is independence between every pair of features of the input given its classification, tends to perform quite well.

7 Derivation of Bayes' Theorem

We can start off with one of the basic probability equalities:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

This reads: the conditional probability of event A given that event B has occurred is equal to the probability of both event A and event B occurring divided by the probability of event B. What we want to find is a relationship between $P(A|B)$ and $P(B|A)$. You can imagine that finding this relationship would be useful in machine learning. It would be able to calculate the probability of a data sample belonging to a certain class given the features of that data sample (the goal of a classification machine learning model) using the probability that a data sample has certain features, given that it belongs to a particular class (something we can calculate from training data).

So, getting back to the equation, if we multiply both sides by $P(B)$, we get:

$$P(A|B)P(B) = P(A \wedge B)$$

Note that $P(A \wedge B) = P(B \wedge A)$, and furthermore, $P(B \wedge A) = P(B|A)P(A)$. Thus,

$$P(A|B)P(B) = P(B|A)P(A)$$

Dividing both sides by B, we arrive at Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Let us consider an example. We want to determine if a specific applicant will receive a job they apply for, given data on previous applicants who applied for the same job, with a discrete feature of the applicant: if they have a college degree. In this case, the probability that they get the job given their degree status = (the probability that previous applicants who got the job had a college degree) $*$ (the probability any previous applicant got the job) / (the probability any previous applicant had a college degree)

8 Bayes' Theorem

We can apply Bayes' Theorem to machine learning. Specifically, say we are given a data sample with n features, $x_1, x_2, x_3, ..., x_n$. We want to find the probability that this data sample belongs to a particular class y. In other words, we want to calculate $P(y|x_1 \wedge x_2 \wedge x_3 ... \wedge x_n)$. (We can use commas instead of the 'and' symbols for shorthand, writing the probability instead as $P(y|x_1, x_2, x_3, ..., x_n)$). Using a dataset of samples which are each labeled with the class y and features $x_1, ..., x_n$, we want to calculate this probability. In other words, consider the job applicant example from before, except now with the job recruiter considering more features, or more qualities of the applicant. Instead of only considering their college degree status (x_1), they could also start to consider if they have previous job experience (x_2), if they fit in well with the workplace (x_3), if they are a business major (x_4), along with a multitude of other features up to (x_n). We can apply Bayes' Theorem:

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)}$$

The numerator of the right-hand side of the equation can be condensed as $P(A|B)P(B) = P(A, B)$:

$$P(y|x_1, ..., x_n) = \frac{P(x_1, ..., x_n, y)}{P(x_1, ..., x_n)}$$

Using the same law, $P(A|B)P(B) = P(A, B)$, we can decompose the numerator like so:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|x_2, ..., x_n, y)P(x_2, ..., x_n, y)}{P(x_1, ..., x_n)}$$

Then, we can decompose the second factor of the numerator in a similar fashion:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|x_2, ..., x_n, y)P(x_2|x_3, ..., x_n, y)P(x_3, ..., x_n, y)}{P(x_1, ..., x_n)}$$

We can keep decomposing the last factor in the numerator until we arrive at:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|x_2, ..., x_n, y)P(x_2|x_3, ..., x_n, y)...P(x_n|y)P(y)}{P(x_1, ..., x_n)}$$

This is nice, but we usually cannot calculate the numerator directly if we are given an unseen sample. If we could, we would already have a sample in our training dataset with the exact same features $x_1, ..., x_n$. The goal is to be able to arrive at an expression that allows us to handle the general case.

At this point, we will have to make a (somewhat fallacious) assumption about the data in order to arrive at a meaningful expression. The assumption

is that each of the features of the data is independent from every other feature. In other words, the probability that a certain feature takes on a certain value is independent of the values that other features take on. In our example of a job applicant, we would have to assume each quality of the applicant as independent from every other quality. We would then regard, for instance, the applicant's possession of a business major as being independent from whether they have a college degree or not. This is definitely not true for examples such as this one, which is why we refer to the classifier as "naive". However, it turns out that the classifier tends to make fairly accurate predictions, even in situations such as this.

The reason we make this assumption is because we really don't like all the conditionals in each probability. This assumption will make all those conditionals go away. If a feature is independent of every other feature, then we can simply remove the features from the conditional part of the probability. For example, let's look at $P(x_1|x_2, ..., x_n, y)$. Our assumption says that feature x_1 is independent of features x_3, x_4, x_5 and so on. This means that $P(x_1|x_2, ..., x_n, y) = P(x_1|y)$ - we can take out all the other features from the conditional. Moreover, we can do this for every single probability with features in the conditional. This is useful because we can then simplify our equation to:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)P(x_3|y)...P(x_n|y)P(y)}{P(x_1, ..., x_n)}$$

We can use capital pi (product) notation to write the n factors in the numerator:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)}$$

This is the fundamental equation of naive Bayes. Specifically, when we are given a sample with an unknown class and features $x_1, ..., x_n$, we can calculate $P(y|x_1, ..., x_n)$ for each possible class y. The class that returns the highest probability when plugged into the equation would be the correct classification for the sample. Each $P(x_i|y)$, or in our example, probability a previous applicant had a certain feature, given that they eventually got the job, is easily calculable from training data on previous applicants.

Note that $P(x_1, ..., x_n)$ does not depend on y (is constant with varying values of y), so we can simply maximize $P(y)\prod_{i=1}^{n} P(x_i|y)$ with respect to y, which will yield our class prediction.

9 Continuous Features

The expression we derived above will work cleanly for categorical features, as we can fairly simply calculate $P(x_i|y)$ for any specific value of the categorical

feature x_i and specific class y. We can do this by dividing the number of samples in our training dataset with that categorical feature and class y by the number of samples with the class y.

Continuous features are a completely different beast. Working with continuous features, we must define $P(x_i|y)$ with a probability distribution. It is very likely that the samples in our dataset will have continuous features that take on values close to - but not exactly equal to - continuous features we will see in the real world. For example, say we are trying to predict the species of flower based on flower heights. We may have a flower in our training dataset with a height of 30 cm. If we are asked to classify a flower with height 29.5 cm, it is likely that flower will be of the same species as the 30 cm flower in our dataset. Due to the heights not being exactly the same, an issue arises. Since there is no 29.5 cm flower, when we try to calculate $P(x_1|y)$, the probability will turn out to be 0, a pretty bad prediction. The solution is to introduce a probability distribution, which shows that since 29.5 and 30 similar and close heights, it is likely that the 29.5 cm flower is the same species as the 30 cm flower.

One way to define this probability distribution is with a Gaussian distribution. If the training data has a continuous feature x_i, we will separate the training data samples based on its class y. For each set of samples with class y, we must calculate the mean μ_y and the variance σ_y^2. We can now define $P(x_i|y)$ as:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{\frac{-(x_i-\mu_y)^2}{2\sigma_y^2}}$$

This equation might seem intimidating at a first glance, but we are only plugging our calculated μ_y and σ_y^2 into the standard Gaussian probability density function. This allows us to solve the previously defined maximization problem to calculate our class prediction. Assuming a normal distribution may not always be the best answer compared other probability distributions we can define to increase the accuracy of our model, but for the sake of simplicity, the normal distribution serves as a decent approximation.

10 References

10.1 Images

- Fig. 1: https://commons.wikimedia.org/wiki/File:KnnClassification.svg