Supervised Learning - Naive Bayes & Support Vector Machines

MInDS @ Mines

In this lecture we continue looking at supervised learning methods. We'll cover Naive Bayes and Support Vector Machines (SVMs) which are both popular methods for varying applications. We also cover the kernel method which is an effective method when applied to data of particular patterns. SVMs coupled with the kernel method are powerful.

Naive Bayes

Naive Bayes is a modeling method that learns from the conditional probabilities that exist between feature values and the resulting target value. To go into how that works, we should first discuss some background about probabilities. A probability is a value between zero and one that denotes the chance of a certain event happening. For example, a fair coin can be heads or tails at equal chance, and there is no other option, which means that the probability of heads is 0.5 and tails is 0.5. The sum of the probabilities of mutually exclusive events, where only one of which can occur at a given point, is equal to 1. Now let's take a look at conditional dependence. *Conditional dependence* means that the probability of an event changes based on another event. *Independence* means that the probability of an event is not affected by the state of another event. For a fair coin, each toss is independent of the previous ones.

Let's go over an example of this using a bag of marbles with 8 blue marbles and 2 orange marbles where we blindly select one of the marbles. The probability of selecting a blue marble, $P(\text{Select} = \text{blue}) = \frac{8}{8+2} = 0.8$, and selecting an orange marble, $P(\text{Select} = \text{orange}) = \frac{2}{8+2} = 0.2$. If after selecting a marble we return it back to the bag then the next turn of selection is independent from this one's and uses the same probabilities. If we select a marble and remove it from the bag then the probabilities have changed. We represent a conditional probability using the $|$ symbol. For example, the probability of selecting a blue marble given that we previously selected a blue marble is, $P(\text{Select} = \text{blue} \mid \text{Previous} = \text{blue})$. If we are removing marbles then $P(\text{Select} = \text{blue} \mid \text{Previous} = \text{blue}) = \frac{7}{7+2} = 0.78$. If we are returning marbles then $P(\text{Select} = \text{blue} \mid \text{Previous} = \text{blue}) = \frac{8}{8+2} = 0.8 = P(\text{Select} = \text{blue})$. When two events, A and B, are independent, their conditional probabilities are equal to their original probabilities, $P(A|B) = P(A)$. To simplify calculations with probabilities, you can always use Bayes' Rule,

$$P(A \mid B) = \frac{P(A)\, P(B|A)}{P(B)}. \tag{1}$$

Bayes' Rule allows us to mathematically represent conditional dependence in data. We can use it to understand how certain features affect a

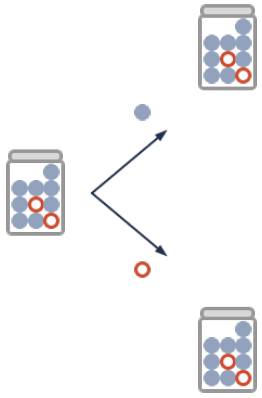

Figure 1: Marbles problem when returning selected marbles.

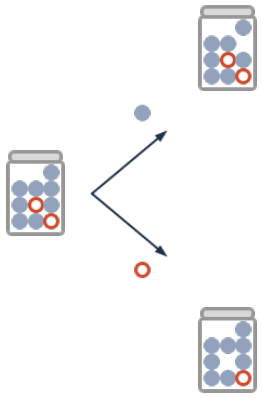

Figure 2: Marbles problem with removal of selected marbles.

If we have an understanding of an event A's occurrence, the probability that it occurs, $P(A)$, is the *prior*. If we determine the existence of another event, B, that may impact A, the conditional probability of A given B, $P(A|B)$, is the *posterior*.

target. The Naive Bayes method uses this concept to create a model for our data. It calculates the probabilities of an input being a particular class given the probabilities of all the inputs being each of the classes.

If we apply Bayes' Rule to this problem we get,

$$P(\text{Class} = i \mid x_1, \ldots, x_n) = \frac{P(\text{Class} = i)\, P(x_1, \ldots, x_n \mid \text{Class} = i)}{P(x_1, \ldots, x_n)}. \quad (2)$$

We will examine a classification problem with k classes where each data item has n features. If we assume that all the features are independent then the probability of a particular feature value resulting in a class is equal to the probability of that feature resulting in the class given all the other features, $P(x_a \mid \text{Class} = i, x_1, \ldots, x_{a-1}, x_{a+1}, \ldots, x_n) = P(x_a \mid \text{Class} = i)$. This assumption is what makes the Naive Bayes method naive.

We can then update the probability of a class given an input as,

$$P(\text{Class} = i \mid x_1, \ldots, x_n) = \frac{P(\text{Class} = i) \prod_{a=1}^{n} P(x_a \mid \text{Class} = i)}{P(x_1, \ldots, x_n)}. \quad (3)$$

From our training data, we can calculate the probability of a class as the proportion of the data that has that class. For the probability of a feature given the class, we can use a variety of methods to calculate them. The value for the probability of a given feature vector is a constant.

From there, we can determine the appropriate class for a particular input vector as,

$$\text{Class} = \arg\max_i P(\text{Class} = i) \prod_{a=1}^{n} P(x_a \mid \text{Class} = i) \quad (4)$$

Three common variations of Naive Bayes (NB) are the Gaussian NB, Multinomial NB, and Bernouli NB. Each of these uses the respective distribution to estimate the probability of a feature given the class.

Naive Bayes variations:

$$P(x_a \mid \underset{Gaussian}{\text{Class} = i}) = \frac{\exp \frac{(x_a - \mu_{a,i})^2}{2\sigma_{a,i}^2}}{\sqrt{2\pi\sigma_{a,i}^2}},$$

where $\sigma_{a,i}$ and $\mu_{a,i}$ are the standard deviation and mean of feature a for class i.

$$P(x_a \mid \underset{Multinomial}{\text{Class} = i}) = \frac{\sum_{\mathbf{x} \in I} x_a + \alpha}{\sum_{a=1}^{n} \sum_{\mathbf{x} \in I} x_a + \alpha\, n},$$

where I is the set of feature vectors of class i, and α is a smoothing parameter.

$$P(x_a \mid \underset{Bernouli}{\text{Class} = i}) = \frac{\sum_{\mathbf{x} \in I} x_a}{m_i} x_a + (1 - \frac{\sum_{\mathbf{x} \in I} x_a}{m_i})(1 - x_a),$$

where all n features are boolean values, I is the set of feature vectors of class i, and m_i is the number of vectors in class i.

Support Vector Machines

Support Vector Machines (SVMs) are another popular method used in machine learning. For a 2-dimensional, 2-class classification problem, SVMs find the line that maximizes the separation between the points of each class. The distance between the two nearest points classified one way or the other is the *margin*. Many lines can separate the sets of points but the goal is to find the line with the largest margin. The points closest to the separating line are *support vectors*.

The output from SVM is the label of a particular point as $\mathbf{w}^T\mathbf{x} + b$. We also get three lines, one where the label is unknown (0) at the line equidistant from the two classes, and two lines where we know the labels are one class or the other (1, -1). The three lines are $\mathbf{w}^T\mathbf{x} + b = 0$, $\mathbf{w}^T\mathbf{x} + b = 1$, and $\mathbf{w}^T\mathbf{x} + b = -1$. If we select one support vector from each class, $\mathbf{x}_+, \mathbf{x}_-$, the margin is the difference between the two points.

In the generalized case of n-dimensional problems, SVMs find the hyperplane that maximizes the separation between the points of each class. A *hyperplane* in an n-dimensional space is a subspace in $n - 1$ dimensions. In 2D, a hyperplane is a line, in 3D, a hyperplane is a plane and in 4D, a hyperplane is a volume.

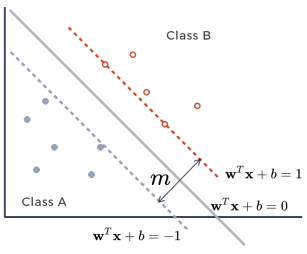

Figure 3: Geometric display of SVM's separating line. m is the margin.

We can state the two support vectors as,

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_+ + b &= 1, \\ \mathbf{w}^T\mathbf{x}_- + b &= -1. \end{aligned} \tag{5}$$

Now let's calculate the margin's distance. Recall that the distance, $||\mathbf{d}||_2$ between a point $\mathbf{x}_i$ and a hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$, is

$$||\mathbf{d}||_2 = \frac{|\mathbf{w}^T\mathbf{x}_i + b|}{||\mathbf{w}||_2}. \tag{6}$$

The margin is therefore the distance from $\mathbf{x}_+$ to the hyperplane plus the distance from $\mathbf{x}_-$ to the hyperplane,

$$\text{margin} = \frac{|\mathbf{w}^T\mathbf{x}_+ + b|}{||\mathbf{w}||_2} + \frac{|\mathbf{w}^T\mathbf{x}_- + b|}{||\mathbf{w}||_2} = \frac{2}{||\mathbf{w}||_2}. \tag{7}$$

We want to maximize the margin for the model, but since we prefer to write objective functions as minimization problems, we can minimize its reciprocal. It also turns out that if we square the ℓ_2-norm, we can solve the problem using straightforward quadratic programming methods. The basic SVM objective function is,

$$\begin{aligned} &\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||_2^2 \\ &s.t. \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall\, i, \end{aligned} \tag{8}$$

where $y_i \in [1, -1]$ determines the class point i belongs to.

Based on this approach, we ensure that the margin separates all points and does not allow any room for error. A *hard margin* does not allow any room for error and this leads to overfitting the data. If we want the margin to allow some error then it becomes a *soft margin* and we use a penalty parameter, C, to determine the softness of the margin. The idea here is that we will allow the model a certain amount of error based on the wrong points it classifies and how wrong these points are. We determine the error for a point using the distance to its class line and we use the C value as a weight to multiply by the sum of all the distances of the wrong points. A higher value for C will result in a "harder" margin and a lower value will result in a "softer" margin.

We update the objective function of SVM to,

$$\begin{aligned} &\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{n}\xi_i \\ &s.t. \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0 \qquad \forall\, i, \end{aligned} \tag{9}$$

where ξ_i is the error for point i, and C is the penalty tuning parameter.

SVMs focus on extreme points that are on the class boundary. This means that they don't necessarily need a large amount of data.

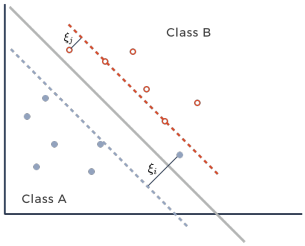

Figure 4: Example of soft margin SVM allowing misclassification which results in better generalization.

Multi-Class

To apply SVMs to more than 2 classes, one approach is to train k SVMs, where k is the number of classes. We train the i^{th} SVM to classify whether a point is of class i or of any of the other classes. We can then use all the hyperplanes of the trained models to classify any given point. This approach is called one versus all (or one versus rest) and we can use it with any binary classifier other than SVMs as well. In each model, we likely have an imbalanced dataset since we are training this class against all others.

Another approach is to train $\frac{k(k-1)}{2}$ classifiers, where we train each classifier on data from two classes and ignore the rest. This approach is called one vs one and can also be used with any other binary classifier.

One vs all will train fewer classifiers that are imbalanced and one vs one will train more classifiers on smaller subsets of data that are likely more balanced.

Regression

To use SVMs for regression we maintain the same objective function but change the constraints to match a continuous value. y_i is no longer the label but the exact value of the target and we no longer use 1 as the inequality constraint but instead use a limit on the acceptable error, ϵ.

The objective function for support vector machines when used for regression is,

$$\begin{aligned} &\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{n}\xi_i \\ &s.t. \quad |y_i - (\langle \mathbf{w}, \mathbf{x}_i\rangle + b)| \leq \epsilon + \xi_i \qquad , \epsilon, \xi_i \geq 0 \forall\, i, \end{aligned} \tag{10}$$

where ϵ is the acceptable error boundary.

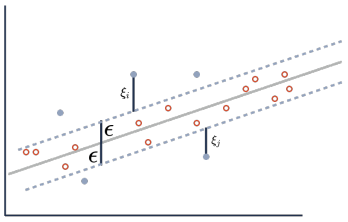

Figure 5: Visualization of SVM when used for regression.

The Kernel Method

SVMs are powerful as they are but a way to make them more effective for particular datasets is to use the kernel method. Everything we've discussed so far assumes that the data is linearly separable (classification) or linearly representable (regression). If the data is not so, we can apply the kernel method to transform it to a new space that is linearly separable or representable. For all the above objective functions, we replace $\mathbf{x}$ with $\phi(\mathbf{x})$ where $\phi(\mathbf{x})$ defines a transformation that occurs to any given point into the new space.

A kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is one that transforms $\mathbf{x}_i$ and $\mathbf{x}_j$ into a new space by taking the inner product of their transformations,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j). \tag{11}$$

A simple example of this is data that appears as 2 concentric circles where the inner circle near the origin is of one class and the outer one is of another class. These two classes in the original space are not linearly separable. We can apply a transformation to the data and the result becomes linearly separable.

Popular kernel methods include,

$$\underset{Polynomial}{K(\mathbf{x}_i, \mathbf{x}_j)} = (\mathbf{x}_i^T\mathbf{x}_j + 1)^d,$$

with degree d,

$$\underset{RBF}{K(\mathbf{x}_i, \mathbf{x}_j)} = \exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}),$$

with width σ,

$$\underset{Sigmoid}{K(\mathbf{x}_i, \mathbf{x}_j)} = \tanh(\kappa\mathbf{x}_i^T\mathbf{x}_j + \theta),$$

with parameters κ and θ.

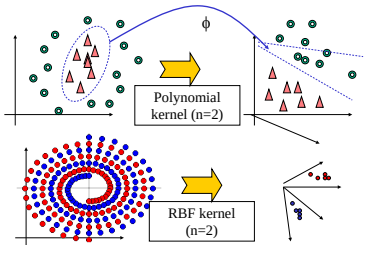

Figure 6: Examples of kernel methods