# 1 Naive Bayes

The binary Naive Bayes classifier has interesting connections to the logistic regression algorithm. In this exercise, you will derive the parametric form of the Naive Bayes classifier under certain assumptions and show that the likelihood function implied by the Gaussian Naive Bayes classifier for two classes is identical in form to the likelihood function for logistic regression. In the second part, you will derive the parameter estimation for the Naive Bayes algorithm.

**1.1 Parametric form of Naive Bayes with Gaussian Assumption** Suppose $X = \{X_1, \ldots, X_D\}$ is a continuous random vector in $\mathbb{R}^D$ representing the features and $Y$ is a binary random variable with values in $\{0, 1\}$ representing the class labels. Let the following assumptions hold:

- The label variable $Y$ follows a Bernoulli distribution, with parameter $\pi = P(Y = 1)$.
- Each feature $X_j$, we have $P(X_j|Y = y_k)$ follows a Gaussian distribution of the form $\mathcal{N}(\mu_{jk}, \sigma_j)$.

Using the Naive Bayes assumption that states "*for all $j' \neq j$, $X_j$ and $X_{j'}$ are conditionally independent given $Y$*", compute $P(Y = 1|X)$ and show that it can be written in the following form:

$$P(Y = 1|X) = \frac{1}{1 + \exp(-w_0 + \mathbf{w}^\top \mathbf{X})}.$$

Specifically, you need to find the explicit form of $w_0$ and $\mathbf{w}$ in terms of $\pi$, $\mu_{jk}$, and $\sigma_j$, for $j = 1, \ldots, D$ and $k \in \{0, 1\}$.

**1.2 Parameter Estimation for Naive Bayes with Gaussian Assumption** Suppose a training set with $N$ examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$ is given, where $\mathbf{x}_i = (x_{i1}, \cdots, x_{iD})^\top$ is a $D$-dimensional feature vector, and $y_i \in \{0, 1\}$ is its corresponding label. Using the assumptions in 1.1 (not the result), provide the maximum likelihood estimation for the parameters of the Naive Bayes with Gaussian assumption. In other words, you need to provide the estimates for $\pi$, $\mu_{jk}$, and $\sigma_j$, for $j = 1, \ldots, D$ and $k \in \{0, 1\}$.

# 2 Nearest Neighbor

Given a training set with $N$ examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$, where $\mathbf{x}_i = (x_{i1}, \cdots, x_{iD})^\top$ is a $D$-dimensional feature vector, and $y_i \in \{1, \cdots, C\}$ is its corresponding label. The nearest neighbor classifier classifies a new example based on its distance to all training examples. Typically, the Euclidean distance is used as the distance metric

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left( \sum_{d=1}^{D} (x_{id} - x_{jd})^2 \right)^{1/2},$$

which assumes that each feature contributes the same to the distance (the features are often normalized before computing the distance). This assumption, however, is often suboptimal in practice.

For example, some features may be noisy and may lead to poor distance measure. Here we introduce feature weights into the distance metric, and consider learning the weights from the training data. Let $w_d \in [0, 1]$ be the weight of the $d$th feature, then the weighted distance be

$$d_{\mathbf{w}}(\mathbf{x}_i, \mathbf{x}_j) = \left( \sum_{d=1}^{D} w_d (x_{id} - x_{jd})^2 \right)^{1/2}, \quad 0 \leq w_d \leq 1,$$

(a) Assume the feature dimensionality $D$ is 3. One way to learn the optimal feature weights is to do an exhaustic search. To enable the search, we often discretize the weights so that the search space is finite. Describe the procedure to search for the optimal feature weights.

(b) What if the feature dimensionality is very high, say $D = 10000$, does the approach in (a) still work? If not, can you propose a new approach? (Note: your approach does not need to be globally optimal).

## 3 Logistic Regression

Consider binary logistic regression model, where the training samples are *linearly separable*.

(a) Given $n$ training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)$, please write down the negative log likelihood (as loss function):

$$\mathcal{L}(\mathbf{w}) = -\log \left( \prod_{i=1}^{n} P(Y = y_i | \mathbf{X} = \mathbf{x}_i) \right).$$

(b) Is this loss function convex? Provide your reasoning.

(c) Show that the magnitude of the optimal $\mathbf{w}$ can go to infinity when the training samples are *linearly seperable*.

(d) A convenient way to prevent the numerical instability issues is to add penalty term to the likelihood function as follows:

$$\mathcal{L}(\mathbf{w}) = -\log \left( \prod_{i=1}^{n} p(Y = y_i | X = x) \right) + \lambda \|\mathbf{w}\|_2^2, \tag{1}$$

where $\|\mathbf{w}\|_2^2 = \sum_i w_i^2$ and $\lambda > 0$. Please compute the gradient respect to $w_i$, i.e. $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_i}$.

(e) Show that the problem in Eq. (1) has a unique solution.

## 4 Decision Tree

(a) Suppose you want to grow a decision tree to predict the *accident rate* based on the following accident data which provides the rate of accidents in 100 observations. Which predictor variable (weather or traffic) will you choose to split in the first step to maximize the information gain?

| Weather | Traffic | Accident Rate | Number of observations |
|---|---|---|---|
| Sunny | Heavy | High | 23 |
| Sunny | Light | Low | 5 |
| Rainy | Heavy | High | 50 |
| Rainy | Light | Low | 22 |

(b) Suppose in another dataset, two students experiment with decision trees. The first student runs the decision tree learning algorithm on the raw data and obtains a tree $T_1$. The second student, normalizes the data by subtracting the mean and dividing by the variance of the features. Then, he runs the same decision tree algorithm with the same parameters and obtains a tree $T_2$. How are the trees $T_1$ and $T_2$ related?

(c) In training decision trees, the ultimate goal is to minimize the classification error. However, the classifaction error is not a smooth function; thus, several surrogate loss functions have been proposed. Two of the most common loss functions are the *Gini index* and *Cross-entropy*, see [MLaPP, Section 16.2.2.2] or [ESL, Section 9.2.3] for the definitions. Prove that, for any discrete probability distribution $p$ with $K$ classes, the value of the Gini index is less than or equal to the corresponding value of the cross-entropy. This implies that the Gini index is a better approximation of the misclassification error.

*Definitions*: For a $K$-valued discrete random variable with probability mass function $p_i, i = 1, \ldots, K$ the Gini index is defined as: $\sum_{k=1}^{K} p_k(1 - p_k)$ and the cross-entropy is defined as $-\sum_{k=1}^{K} p_k \log p_k$.

# 5 Programming

In this assignment, you will experiment with four commonly used classification algorithms on a real-world dataset. You will use MATLAB's functions (use the R2013b version available from software.usc.edu) to experiment with Decision Tree and Logistic Regression, but you need to implement $k$-nearest neighbor and Naive Bayes algorithms by yourself. You are NOT allowed to use any related Matlab toolbox functions for $k$NN and Naive Bayes (e.g. `knnclassify`, `knnsearch`). Below, we describe the steps that you need to take to accomplish this programming assignment.

**Dataset**: We have preprocessed the *Car Evaluation Dataset* from UCI's machine learning data repository. The training/validation/test sets are provided along with the assignment in Blackboard as `cars_train.data`, `cars_valid.data`, and `cars_test.data`. For a description of the dataset, please refer to http://archive.ics.uci.edu/ml/datasets/Car+Evaluation.

Please follow the steps below:

**5.1 Data Inspection** The first step in every data analysis experiment is to inspect the datasets and make sure that the data has the appropriate format. You will find that the features in the provided dataset are categorical. However, some of the algorithms require the features to be real-valued numbers. To convert a categorical feature with $K$ categories to real-valued number, you can create $K$ new binary features. The $i$th binary feature indicates whether the original feature belongs to the $i$th category or not.

**5.2 Implement Naive Bayes** Please fill in the function `naive_bayes` in `naive_bayes.m` file (in Blackboard). The inputs of this function are training data and new data (either validation or testing data). The function needs to output the accuracy on both training and new data (either validation or testing). Note that some feature values might exist in the validation/testing data, but do not exist in the training data. In that case, please set the probability of that feature value to a small value, for example, 0.1.

**5.3 Implement *k*NN** Please fill in the function `knn_classify` in `knn_classify.m` file (in Blackboard). The inputs of this function are training data, new data (either validation or testing data) and $k$. The function needs to output the accuracy on both training and new data (either validation or testing).

**5.4 Performance Comparison** Compare the four algorithms ($k$NN, Naive Bayes, Decision Tree, and Logistic Regression) on the provided dataset.

***k*NN:** Consider $k = 1, 3, 5, \cdots, 23$. For each $k$, report the training, validation and test accuracy. When computing the training accuracy of $k$NN, we use leave-one-out strategy, i.e. classifying each training point using the remaining training points. Note that we use this strategy only for $k$NN in this assignment.

**Decision Tree:** Train decision trees using function `ClassificationTree.fit` or `fitctree` in Matlab. Report the training, validation and test accuracy for different split criterions (*Gini index* and *cross-entropy*, using the SplitCriterion attribute), by setting the minimum number of leaf node observations to $1, 2, \cdots, 10$ (using the MinLeaf attribute). So basically, you need to report the results for $2 \times 10 = 20$ different cases. When training decision trees, please turn off pruning using the `Prune` attribute.

**Naive Bayes:** Report the training, validation and test accuracy.

**Logistic Regression:** Train multi-class logistic regression using the function `mnrfit` in Matlab. Report the training, validation and test accuracy.

**5.5 Decision Boundary** In this step, you need to apply $k$NN on the `boundary.mat` dataset (provided in Blackboard) which is a binary classification dataset with only two features. You need to run $k$NN with $k = 1, 5, 15, 20$ and examine the decision boundary. A simple way to visualize the decision boundary, is to draw 10000 data points on a uniform $100 \times 100$ grid in the square $(x, y) \in [0, 1] \times [0, 1]$ and classify them using the $k$NN classifier. Then, plot the data points with different markers corresponding to different classes. Repeat this process for all $k$ and discuss the smoothness of the decision boundaries as $k$ increases.

**Submission Instruction:** You need to provide the followings:

- Provide your answers to problems 1-4, 5.4, and 5.5 in hardcopy. The papers need to be stapled and submitted to the CS front desk.
- Submit the source code of the two functions which are required in 5.2 and 5.3 via Blackboard. The only acceptable language is MATLAB.

**Collaboration** You may collaborate. However, collaboration has to be limited to discussion only and you need to write your own solution and submit separately. You also need to list with whom you have discussed.