WiFi-Task

Goutam Y G

August 23, 2017

We have N=63134 instances of feature vectors $x_i \in R^{137}$ and labels $y_i \in R$. (Though labels have only finite number of discrete states, we assume that $Y \in R$ for simplicity)

As first step, we append 1 to each feature vector. It enables to learn models with higher complexity, i.e. affine models. Without the bias term, one can learn only linear models. (which pass through origin)

Training

We use following models for the task:

- **Linear Regression:** In this method, we learn model parameters w such that $Xw \approx Y$. Model parameters are obtained by minimizing $E(w) = ||Y - Xw||^2$

  Closed form solution would be, $w = (X^T X)^{-1}(X^T Y)$

  Somtimes, directly solving for w is computationally expensive because of matrix inversion. If $X^T X$ is not invertible, this formula cannot be used. In these cases, we can use iterative schemes like gradient descent. Python's *sklearn* library has inbuilt functions to minimize $E(w)$ and learn model parameters w.

- **Ridge Regression:** We use same error measure to learn the parameters, but with a $L2$ regularization. It helps to avoid overfitting, but requires tuning of regularization parameter.

- **Logistic Regression:** We minimize the logistic loss function with $L2$ regularization to learn the weights of linear classifier.

- **SVM:** We minimize hinge loss function with $L2$ regularizer to learn the weights of linear classifier.

- **k-Nearest Neighbor(kNN):** It is the simplest algorithm in the sense that, it requires no training. A sample in test dataset is classified based on class labels of k nearest neighbors from training data. (with suitable methods to handle ties) The decision boundary is non-linear and can be useful if datapoints of different classes are clustered and linearly non-separable.

Algorithm Evaluation using test data

- One simple method is compute **accuray**, i.e.

  $$\text{accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of datapoints in test data}}$$

  For a n-class classification problem, we expect an accuracy which is significantly above $\frac{1}{n}$. (better than random guess)

- Another approach would be to compute the **confusion matrix** (call it C). It is of dimension $n * n$ where n is the number of classes. Digonal elements C represent how many times an element in class i is correctly classified as class i. Entry C_{ij} represents how many times a sample in class i is misclassified as class j. It gives a better picutre about misclassification esp. to understand about classes which cause the confusion.

Please read this section before running the code

The experiments are run on a machine with Intel $i5$ processor and 8GB RAM.

- Choose a model by uncommenting the corresponding code

- Logistic regression, SVM are *extremely* slow during training (because train multiple classifiers for multiclass classification)

- Linear Regression, Ridge Regression have least training time, but bad accuracy (measured on training data)

- kNN takes a lot of time during testing. (because of computing distances from all training points for classification)