Solution to HMS Challenge #1: Minority Variants

Chun-Sung Ferng (handle: csferng)

1 Data Preparation

The training data contains more than 16M reads, while only 400K are the interest of this problem (true minority variants and fake constants). My first step is to extract these 400K reads and separate them into two parts, one for training (referred as *training data* below) and the other for offline validation. This is done through a python script `data.py`.

In my solution, about one fourth of the reads are chosen for validation randomly. Although it is mentioned in the problem statement that the frequency of minority variants in testing data would be less than or equal to 6.5%, I extracted all true minority variants with frequency lower than 7%. In following sections, *label* would be used to denote the type of a read, where true minority variants have label 1, and fake constants have label 0.

2 Model Training

My solution to this problem is to build a Random Forest [1] model using the training data. A Random Forest model consists of a number of decision trees. (Here 50 trees are used.) Each decision tree is trained on a bootstrap sample of the training data. A bootstrap sample generates the same number of reads as training data by sampling uniformly and with replacement from training data.

2.1 Feature Expansion

Before training the model, I expanded the 3 metrics of each training read to 8 features using logarithm and multiplications. Let r_i, k_i, m_i be read quality, K-mer abundance and mapping score of the i-th training read. The following 8 features are used for training:

$$r_i^2 k_i m_i^2 \ , \ r_i^2 k_i^2 m_i \ , \ k_i m_i \ , \ r_i k_i m_i \ , \ r_i k_i^2 m_i \ , \ r_i k_i^2 m_i^2 \ , \ \frac{r_i m_i}{(\log(k_i))^2} \ , \ \frac{r_i m_i}{-\log(k_i)}$$

$\mathbf{x}_i$ would be used for indicating these 8 features, called as *feature vector*. These formulas are selected manually based on the performance of trained model on offline validation.

2.2 Decision Tree Training

A decision tree is constructed by splitting the training data into two subsets based on a *good criterion*. This process is repeated recursively on each subset until a *stopping condition* is satisfied.

Good splitting criterion This is the most complicated part of my solution. The splitting criterion I used is in the form $\text{sign}(\mathbf{w} \cdot \mathbf{x} - b)$, where $\text{sign}(\cdot)$ extracts the sign of its argument. This criterion splits feature vectors $\mathbf{x}$ into two subsets, $\mathbf{w} \cdot \mathbf{x} - b < 0$ and $\mathbf{w} \cdot \mathbf{x} - b \geq 0$. To find good $(\mathbf{w}, b)$ which fit the objective (area under the ROC curve, or AUC), the following procedure is used at each internal node.

1. Generate N random vectors $\mathbf{w}_1, \cdots, \mathbf{w}_N$. To simplify the split, each $\mathbf{w}_i$ is limited to have K non-zero components.
2. Update each $\mathbf{w}_i$ by *stochastic cyclic coordinate ascent* (SCCA) for T iterations.
3. Compute b_i for each $\mathbf{w}_i$ which maximizes AUC when using $\text{sign}(\mathbf{w}_i \cdot \mathbf{x} - b_i)$ as confidence of $\mathbf{x}$.
4. Choose the $(\mathbf{w}_i, b_i)$ which reaches the maximum AUC in the previous step.

In step 2, SCCA is applied on each $\mathbf{w}_i$ to improve AUC assuming that $\mathbf{w} \cdot \mathbf{x}$ is used as confidence of $\mathbf{x}$. This confidence function is slightly different to the one above, but I assume the performance of these two have strong connections. Here is the procedure of SCCA:

1. Randomly draw P pairs $(\mathbf{x}_{p1}, \mathbf{x}_{p0})$ where $\mathbf{x}_{p1}$ has label 1 and $\mathbf{x}_{p0}$ has label 0.
2. Calculate $d_p = \frac{\mathbf{w}_i \cdot (\mathbf{x}_{p0} - \mathbf{x}_{p1})}{\mathbf{x}_{p1}[j] - \mathbf{x}_{p0}[j]}$ and $v_p = \text{sign}(\mathbf{x}_{p1}[j] - \mathbf{x}_{p0}[j])$ for a coordinate j where $\mathbf{w}_i[j] \neq 0$.
3. Compute $d^* = \arg\min \sum_{p:d_p<d^*} v_p$, which can be done via sorting and linear search.
4. Add d^* into $\mathbf{w}_i[j]$.
5. Do step 1 to 4 for all the other j where $\mathbf{w}_i[j] \neq 0$.
6. Repeat step 1 to 5 T times.

This method is cyclic because it updates each non-zero component of $\mathbf{w}_i$ once per iteration. And it is stochastic since it only considers P pairs for each update, instead of all possible pairs (which may be computationally infeasible). For the submitted model, $N = 8$, $K = 3$, and $T = 3$ are used. P is set to the minimum between the number of pairs in the subset and half of the size of total training data.

Stooping condition A leaf node is created if any of the following three conditions is satisfied. This leaf node later on will predict a constant, which is the average of the labels in the subset of training data at this node.

- All reads in this subset have the same label.
- The size of this subset is less than 100.
- The depth of the tree reaches a specified maximum allowed depth (set to 7).

2.3 Encoding Trees to String

After training, the decision trees are encoded to a string and saved for later testing. The encoding of a leaf node only contains its prediction value. An internal node which uses $(\mathbf{w}, b)$ for splitting is encoded in this format:

$$b\texttt{,}j_1\texttt{:}\mathbf{w}[j_1]\texttt{,}\cdots\texttt{,}j_K\texttt{:}\mathbf{w}[j_K]\texttt{(}\textit{child 1}\texttt{)(}\textit{child 2}\texttt{)}$$

Note that the $\mathbf{w}$ is encoded in sparse format, i.e. only non-zero components are encoded. The encoding of all the decision trees are concatenated into one string.

2.4 Confidence of Testing Reads

To calculate the confidence of testing reads, the first step is to parse the encoded string back to decision trees. Then the feature expansion is done on each testing read as described in Section 2.1. The confidence (raw score) of a testing read is the sum of the prediction values it gets from all decision trees.

To convert raw score into an integer between 0 and 10^9, all testing reads are sorted by their raw score and their ranks in sorted order are taken as the confidence of them. The idea of this conversion is to prevent rounding error.

3 Program Usage

- `data.py`: This python script is for generating training and validation data, as described in Section 1.

  `python data.py` [path/to/inputs/]

  It takes one argument, which is the path to the directory containing `input1.tsv`, `input2.tsv` and `locations.tsv`. It will output two files to the same directory, `train.txt` and `valid.txt`.

- `tree.cpp`: This is the program to train Random Forest model.

  `./tree -data` [path/to/train.txt] `-ntree` M `-mxdep` D `-trail` N `-nfpn` K `-cdit` T

  All parameters besides training data (`-data`) are optional. M is the number of trees to build in the Random Forest, and D is the maximum allowed depth of the decision trees. N, K, and T are the same as described in Section 2.2. The trained and encoded model will be outputted to standard output.

- `tree-test.cpp`: This is the program to perform offline validation.

  `./tree-test` [path/to/valid.txt] [path/to/model]

  This program parses the model, calculates the confidence for each read in `valid.txt` and outputs the AUC score that the model gets.

References

[1] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.