Predicting Water Well Statuses in Tanzania

CS 205 Final Project

Dmitry Vinichenko, Sam Kim, Robert Hoyt

Introduction

"Pump it Up: Data Mining the Water Table" is a charity-motivated competition hosted by Driven Data to help classify water wells in Tanzania [1]. Ground data is supplied as a pair of CSV files with 39 feature observations and the corresponding well class: "functional", "functional needs repair", and "non functional". In this report we describe a method for eliminating irrelevant and redundant categorical features, preprocessing steps to help random forests classify data associated with GPS coordinates of the wells, and a parallelization strategy to expedite generation of large random forests.

Background

Access to clean drinking water is a critical element of infrastructure for developing third-world countries. Tanzania is no exception, and due to widespread poverty and unsuccessful government programs, access to clean drinking water remains a challenge. To help address this issue, Driven Data is working with the Tanzanian Ministry of Water to host the competition "Pump it Up: Data Mining the Water Table." The idea is simple: given 39 different feature observations, is it possible to determine the operating status of a well? For this competition the features are a mix of continuous numerical data (e.g. GPS coordinates) and categorical data (e.g. type of water source).

Approach and Secret Weapons

Our approach to this process is composed of three components: preprocessing categorical features using the Minimal Redundancy - Maximal Relevance method (mRMR), preprocessing GPS coordinate data, training random forest classifiers in parallel on all preprocessed data.

Decision Trees

To tackle the classification challenge we decided to use random forests [2] as implemented in scikit-learn package for Python [3]. Random forest classifiers are ensemble learners composed of many randomized decision trees. Their key advantages for this challenge include unbiased estimates of the generalization error [2], near immunity to irrelevant and correlated features, invariance under monotonic feature transformations, and the ability to simultaneously treat numerical and categorical data.

A decision tree classifier uses training data to construct a series of locally-optimal, if-else decision rules to obtain a unique classification of data. For a given set of features with corresponding observations, all possible partitions for each feature are considered. The optimal choice is the one that maximizes the overall class purity of the leaves, leading to (locally) maximum predictive power. An example is shown in Figure 1. The first decision selects dry vs. not dry wells, leading to the "not dry" leaf with slightly increased purity, and a "dry" leaf with substantial purity. The next optimal decision for each leaf is different, but in both cases maximize the resulting purity of their children. A decision tree is thus built recursively, in our case until the purity cannot be improved.

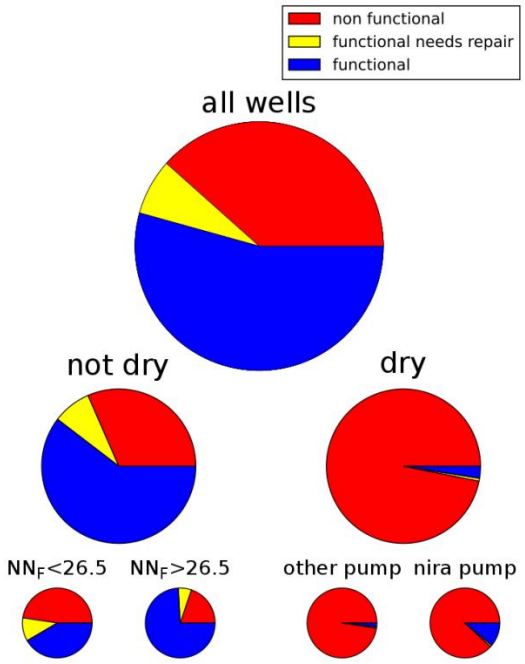

Figure 1. Recursive splits maximize purity. Red – non functional. Yellow - functional needs repair. Blue - functional. NN_F is the number of functional nearest neighbors.

Random Forests

A random forest, using Breinman's original implementation, introduces randomness among trees via bagging, i.e. constructing each tree on a randomly-sampled subset of the training data, and by restricting each node to split on a random subset of the features. Combining these helps remove the high bias inherent in decision trees, and bagging yields unbiased error estimates by calculating the classification error for each sample using the trees that were not trained on it - this is the out-of-bag (OOB) error. Random forests are generally immune to irrelevant features because their corresponding partitions rarely yield higher-purity splits than relevant features. The partitioning process also yields invariance to any transformation that maintains the strict, weak order of feature observations (e.g. constant scaling). This allows the seamless introduction of one-hot encoded features alongside raw numerical data, and removes any need to scale or shift data.

Preprocessing Categorical Features

Our dataset includes 32 categorical features, with 66010 distinct values; moreover, some of the features were apparently redundant. Therefore, we used a systematic multi-step method to choose relevant features and their distinct values.

Filtration of rare values: For each feature a list of frequency of distinct values was created, and the values occuring with frequency lower than certain threshold (1%) were replaced with 'other' in corresponding data entries.

Filtration of irrelevant features and values: Next step was to eliminate the features and distinct values which do not help in classification of the wells. For that, we resorted to theoretical criterion - *mutual information (MI)*. For 2 random variables X and Y with values $\{x_i\}, i = 1 \dots m$, $\{y_j\}, j = 1 \dots m$ and dataset of size N_{tot} one can define outcomes x_i as $p(x_i)$ and y_j as $p(y_j)$,and of simultaneous outcome $p(x_i, y_j)$. With that, MI can be defined as:

$MI(X,Y) = \sum_i \sum_j p(x_i, y_j) \ln \frac{p(x_i,y_j)}{p(x_i)p(y_j)}$.

It is an analogue of correlation for discrete variables, being close to zero for independent features. We used MI between a given feature and the classification labels as a metric for that feature's importance and a criterion for discarding the irrelevant ones. Namely, we defined a feature as irrelevant if its MI was lower than a certain threshold.

Since our version of random forest implementation required vectorization of categorical features into boolean columns, we had to treat features and distinct values on the same footing. For that, for each feature we've sorted the values from most to

least frequent. Then, we've computed the contribution to MI with classification labels for each distinct value, with example shown in Fig. X. The values with MI contribution below the given threshold were replaced with 'other'.

Minimal redundancy - maximal relevance (mRMR) feature filtration: After eliminating obviously irrelevant values and features we had to deal with redundancy of almost equally important features (e.g., "quantity" and "quantity_group"). For that purpose, we found mRMR method [4] being useful. It relies on the concept of "mutual information quotient", which is a ratio of MI of candidate feature with the classification labels c to the average of MI of candidate feature with those already selected.

$$MIQ = \frac{MI(X_c, c)}{\frac{1}{|S|}\sum_S MI(X_c, X_s)},$$

where X_s denotes features selected in the previous steps of the algorithm. In mRMR method, one starts by picking the feature with largest MI with the classification labels and adding it to the set of selected features S. Then, at each step of the algorithm, features are ranked according to mutual information quotient (MIQ), the feature with the largest MIQ is selected, and the MIQs for remaining features are recomputed (since the set of selected features was expanded). The process goes on until largest MIQ for remaining candidate features goes below certain cutoff value. As the procedure proceeds, the features which are both highly informative from classification standpoint and uncorrelated with features already selected are added to the set S. Eventually, candidate features become more and more correlated with the already chosen ones, and the process stops.

Feature vectorization: Finally, for each value in each remaining feature a corresponding column with boolean values showing if entries have this type of value, is added to the data set, and the original features are removed. Another pass of mRMR on the boolean features is made, which in the end leads to about 10 distinct values (out of 66k), which contribute most of the important information for classification.

Preprocessing GPS Data

We discovered that the GPS coordinates for wells are strongly clustered by their classification. In particular, functional and non functional wells typically appear in distinct clusters. A naive forest trained on the raw GPS coordinates, making West-East and North-South decisions only, achieves approximately 70% classification accuracy! This motivated us to engineer the GPS data in an effort to optimize the classification accuracy.

In Breinman's original random forest paper [2] he suggests adding linear combinations of numerical features when the number of features is small to minimize correlation between trees. A linear combination of GPS coordinates corresponds to a *rotation*, allowing trees to make non-axis-aligned boundaries like Northwest-Southeast. Since random forests are invariant to the order of feature columns and scaling by a constant,,only rotation angles in [0, pi/4] are distinct. Our work uses 5 equally-spaced angles in this range. These angles yield irregular polygons in GPS space.

Finally, we considered the close relationship between forests and k-nearest neighbors (kNN). Various methods of weighting neighbors by distance are common and are generally rotationally-symmetric. We decided to add kNN-derived data to the forest's training and testing sets for two reasons. First, these enable nonlinear (in GPS space) and rotationally-symmetric

boundaries to used for decisions. Second, partitions on this kind of data are immediately useful to the forest, e.g. wells with many nearby functional neighbors are likely functional as well. This contrasts with the raw GPS data since clusters are separated only at small length scales, requiring much deeper trees. Each tree was trained on the sum of 1/distance, two sigmoidal functions of distance, and the number of neighbors over each class. The effects are immediately obvious - forests with kNN data have higher OOB scores and exhibit more natural, curved decision boundaries between adjacent clusters.

Performance evaluation

Filtration of rare values: Inside each categorical feature, the values which occur in less than 1% of the total number of data points (59400), were replaced by label 'other'. With such cutoff of 1%, about 260 distinct values remain. A more strict limit could have been imposed, but we wanted to keep more potentially important features for information-theoretic filtration.

Discarding irrelevant features and values: After some testing, for this particular case the optimal threshold was found to be 5% of the maximal MI among the present features, since in our case we did not have features with strongly dominating MI.

mRMR filtration: The MIQ mode was found to be superior, with cutoff value being around 0.7. One could do filtration either twice: before and after vectorization, or only once – after it. In the latter case, more distinct values remain, improving OOB score by about 1%.

GPS rotation and kNN features: Adding several representations of GPS data to the training set proved useful. Table 1 shows the OOB score (total classification accuracy) for several choices of GPS data to include.

We see that adding rotations yields a mildly-useful 0.3% increase to the OOB accuracy, but much better gains are observed by including kNN-derived data. Using a combination of kNN data and rotations yields the highest OOB accuracy of 83.04%, so preprocessing GPS data yields a +2.2% increase in the classification accuracy.0

Number of Trees

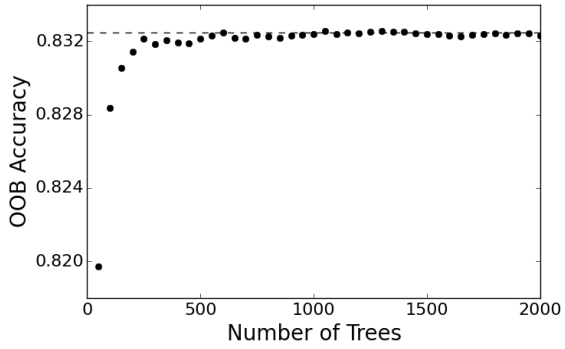

Figure 2. OOB accuracy increases with the number of trees.

Figure 2 shows the asymptotic convergence of the forest's OOB accuracy as the number of trees increases. The OOB accuracy is well-converged at 1000 trees, and since it's derived for each datapoint with about 1/3 of the total number of trees, the forest overall is converged at around 350 trees.

Parallelization

Random forests are embarrassingly parallel because each tree is trained independently. The most straightforward way to implement random forests in parallel is to assign an equal portion of the *n* trees to the *m* cores, so each core calculates *n/m* trees and then send them all back to the root core.

Table 1. Prediction accuracies for different data preprocessing steps.

Data type	no GPS data	raw GPS only	5 total rotations	kNN only	5 rotations & kNN
OOB score	79.25%	80.87%	81.18%	82.53%	83.04%

A more sophisticated approach is to use load-balancing through the master-slave approach, in which the *n* trees are divided into more than *m* groups. The master slave assigns work to any free cores until all *n* trees have been calculated, similar to the MapReduce framework. This is important because the trees do not necessarily take the same amount of time to train, and some cores may fail or become slow.

Figure 3 shows the training time as a function of cores. We see that there is an optimum number of cores - 4 cores without load-balancing, and 16 cores with load-balancing. This is likely due to the communication overhead outweighing the benefits of parallelization. Figure 5 shows the broadcast time of the data as a function of cores. As expected, this increases quickly with the number of cores. There is a big jump going from 8 to 16 cores, which may be due to non-local communication.

Figure 4 shows the training time as a function of trees in a single work-unit for the load-balancing training. There also seems to be an optimum point, due to the balance between communication time and training time.

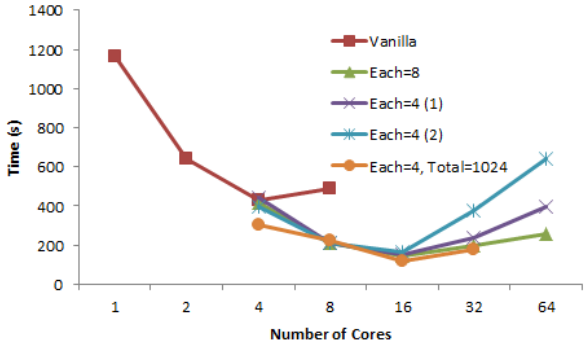

Figure 3. Training time as a function of number of cores, excluding time to broadcast data. Trial number in parentheses. 2048 cores unless otherwise specified. Vanilla refers to non-load-balancing.

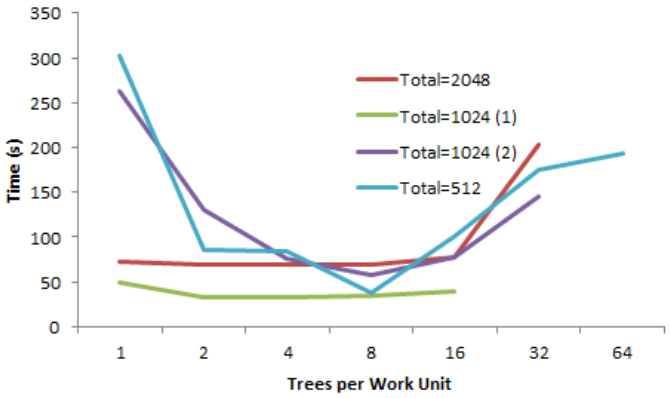

Figure 4. Training time for load-balancing version as a function of number of trees in a single work unit, using 32 cores. Excludes time to broadcast data. Number in parentheses represent the trial.

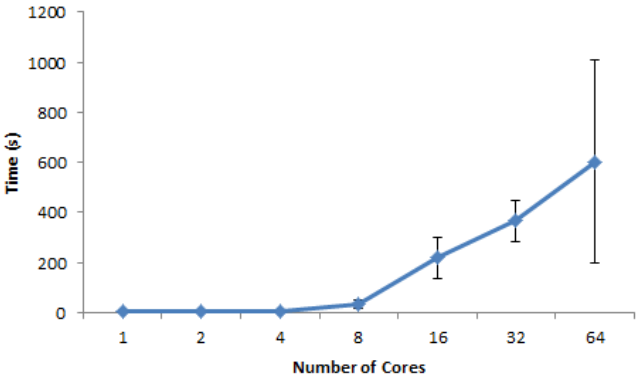

Figure 5. Time to broadcast data as a function of number of cores.

Conclusion

In this report we presented an end-to-end system that preprocesses categorical data using the MRMR method, preprocesses GPS data, and provides data-parallelized random forest classification. Our MRMR system dramatically reduces the number of categorical features that must be one-hot encoded with minimal decrease in accuracy, which should generalize well to any machine learning application which involves substantial amounts of categorical data. In addition, the GPS preprocessing highlights the importance of engineering features for random forests that allow immediate gains in purity. Finally, the parallelization of random forests yields substantial training speedups. These gains could facilitate

extensive parameter sweeps over the various forest options available in scikit-learn, and the data-parallel approach should generalize well to other ensemble-based learning techniques.

References

[1] Pump it Up: Data Mining the Water Table. Driven Data. http://www.drivendata.org/competitions/7/

[2] L. Breinman. "Random Forests". *Machine Learning*, vol. 45, issue 1, pp. 5-32. 2001.

[3] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[4] H.C. Peng, F. Long, C. Ding. "Feature selection based on mutual information: criteria of max-dependency, max-relevance and min-redundancy". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, No. 8, pp. 1226-1238. 2005.