Module 3: Final Project

Water Pump Operational Status Classification

Competitive Data Science Project: Pump it Up: Data Mining the Water Table, hosted by DRIVENDATA

by Steven Contreras

The problem

Context

“Taarifa is an open source platform for the crowd-sourced reporting and triaging of infrastructure related issues. Think of it as a bug tracker for the real world which helps to engage citizens with their local government. We are currently working on an Innovation Project in Tanzania, with various partners.”

- Taarifa

The particular goal is to bring filtered-water, sourced and pumped from local water-sources, to disparate and impoverished geographic locations.

Problem statement

This is a multi-class classification machine learning problem.

The goal is to predict the operating condition of a waterpoint for each record in the dataset, which can be one of the three operational status classes: “*functional*”, “*functional needs repair*”, and “*non functional*”.

Workflow: Adapted from OSEMN

EDA

This phase is geared toward a detailed understanding of all features, including their data-types and special concerns with the specific goal of producing the set of all applicable preprocessing options.

The output of this phase is used as input to the Dynamic Programming algorithm used in the Preprocessing phase.

Preprocessing

This phase executes a Dynamic Programming algorithm to optimize (the “best”) selection of

- Features
- Transformations thereof

… in order to produce models with the highest validation accuracy.

This phase consumes the configuration of applicable preprocessing options discovered in EDA.

The output of this phase is used as input for final transformations used for model building.

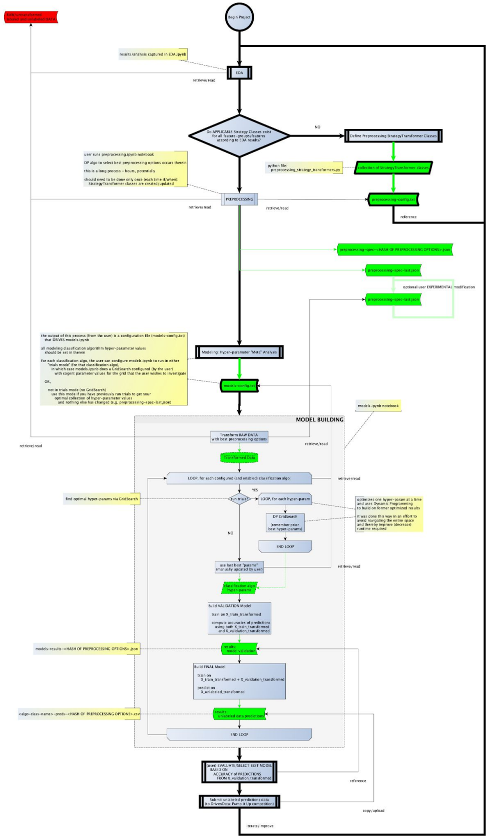
Model Building

This phase focuses on:

- Building an array of models using various classification algorithms
- Optimizing validation accuracy via parameter-tuning
- Building output (CSV) files containing unlabeled predictions produced from those models; these are the deliverables that are submitted to the competition for scoring

This phase consumes the output of the optimized (“best”) selection of features and transformation options output from the Preprocessing phase in order to produce the final transformed data used in model building.

Visualization: Workflow



RAW/untransformed
labaled and unlabeled DATA

Begin Project

results/analysis captured in EDA.ipynb

EDA

retrieve/read

Do APPLICABLE Strategy Classes exist
for all feature-groups/features
according to EDA results?

NO

Define Preprocessing StrategyTransformer Classes

python file:
preprocessing_strategy_transformers.py

collection of StrategyTransformer classes

user runs preprocessing.ipynb notebook

DP algo to select best preprocessing options occurs therein

this is a long process - hours, potentially

should need to be done only once (each time if/when):
StrategyTransformer classes are created/updated

PREPROCESSING

retrieve/read

retrieve/read

preprocessing-config.txt

reference

preprocessing-spec-<HASH OF PREPROCESSING OPTIONS>.json

preprocessing-spec-last.json

optional user EXPERIMENTAL modification

preprocessing-spec-last.json

the output of this process (from the user) is a configuration file (models-config.txt)
that DRIVES models.ipynb

all modeling classification algorithm hyper-parameter values
should be set in therein

for each classification algo, the user can configure models.ipynb to run in either
"trials mode" (for that classification algo),
in which case models.ipynb does a GridSearch configured (by the user)
with cogent parameter values for the grid that the user wishes to investigate

OR,

not in trials mode (no GridSearch)
use this mode if you have previously run trials to get your
optimal collection of hyper-parameter values
and nothing else has changed (e.g. preprocessing-spec-last.json)

Modeling: Hyper-parameter "Meta" Analysis

models-config.txt

models.ipynb notebook

MODEL BUILDING

Transform RAW DATA
with best preprocessing options

retrieve/read

retrieve/read

Transformed Data

LOOP, for each configured (and enabled) classification algo:

retrieve/read

find optimal hyper-params via GridSearch

run trials?

YES

LOOP, for each hyper-param

optimizes one hyper-param at a time
and uses Dynamic Programming
to build on former optimized results

it was done this way in an effort to
avoid navigating the entire space
and thereby improve (decrease)
runtime required

DP GridSearch

(remember prior
best hyper-params)

NO

END LOOP

use last best "params"
(manually updated by user)

retrieve/read

classification algo
hyper-params

Build VALIDATION Model

train on X_train_transformed

compute accuracies of predictions
using both X_train_transformed
and X_validation_transformed

models-results-<HASH OF PREPROCESSING OPTIONS>.json

results:
model validation

Build FINAL Model

train on
X_train_transformed + X_validation_transformed

predict on
X_unlabeled_transformed

<algo-class-name>-preds-<HASH OF PREPROCESSING OPTIONS>.csv

results:
unlabeled data predictions

END LOOP

(user) EVALUATE/SELECT BEST MODEL
BASED ON
ACCURACY of PREDICTIONS
FROM X_validation_transformed

reference

Submit unlabeled predictions data
(to DrivenData: Pump it Up competition)

copy/upload

iterate/improve

View the diagram in your browser here.

Solution

Ensemble (`VotingClassifier`) of classification algorithms: `RandomForestClassifier`, SVM.SVC(RBF Kernel), and `CatboostClassifier`

Competition Leaderboard:

- *Accuracy*: **81.82%**
- *Rank*: **874/9748**

Implementation

ALL PHASES PLAYED EQUALLY IMPORTANT ROLES!

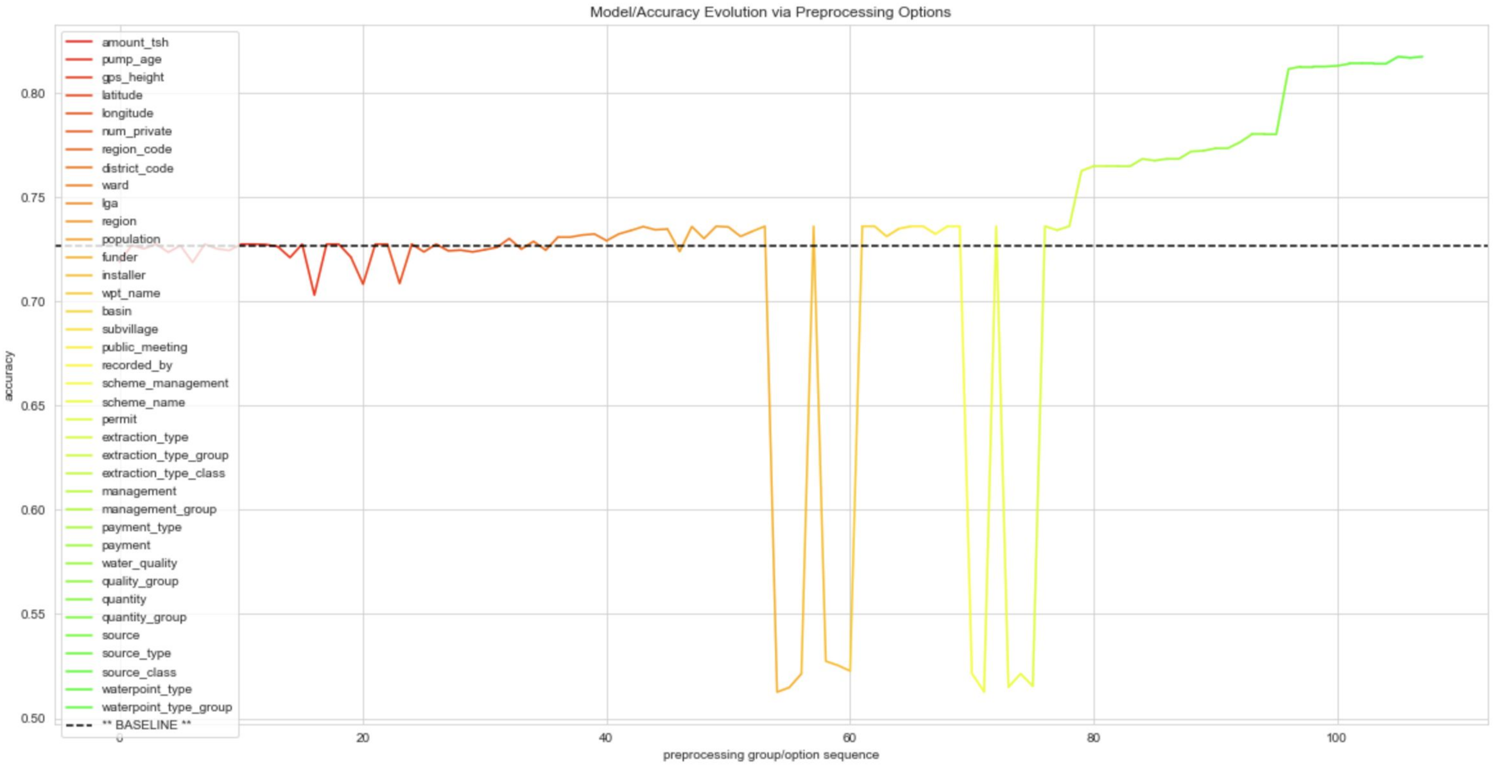
Phase Detail: EDA

- **Foundation and dependency of subsequent phases**
- **Purity of goal**: complete and thorough investigation of EACH feature AND related features (by description… the “kind” of feature… not data type though) to **uncover ALL possible and applicable transformations**; *this required resisting the urge to pollute the EDA process with actual, applied preprocessing and transformations that might be used in final model building*
- Required the **Development and Incorporation of Advanced Techniques for handling High-Cardinality categorical features** (e.g. containing 2000+ “unique” categories)
  - *Target-Encoding*
  - NLP Techniques
    - *TF-IDF Vectorization*
    - Experimental Technique: unsupervised *KMeans Clustering* classification on top of *TF-IDF Vectorization*

Phase Detail: Preprocessing

- **Dynamic Programming algorithm that optimizes feature selection AND the best preprocessing transformation to use**
- **Depends on building a baseline model with a flexible classification algorithm:** XGBClassifier was used for baseline model
- **The best results (optimized set of preprocessing options) depend on a COMPLETE set of applicable transformations for a given feature;** note that this necessitated considering that dropping the feature or “leaving it as is” could be the best option
- **Complexity (big “O”) falls between linear and exponential** (but is not intractable - i.e. VASTLY improves on NP-Hard brute-force) **when there is more than one feature in a feature-group** (related by “kind” of data it represents, provided by feature description) **but is otherwise linear**; thus, **it is imperative to be thorough and at the same time careful to not include extraneous/unnecessary preprocessing options for a given feature**
- **Basis for evaluation: evolution/improvement of validation-set accuracy of baseline model while isolating candidate preprocessing/transformation option**

Visualization: Preprocessing (Optimization)

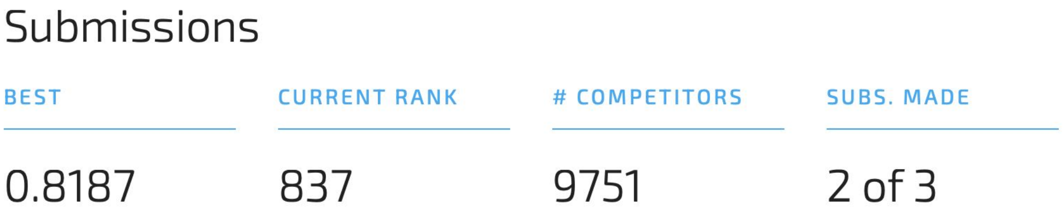


Phase Detail: Model-Building

- **Algorithms considered:**
  - `DecisionTreeClassifier`, `RandomForestClassifier`, `XGBClassifier`, `CatboostClassifier`, `SVM.SVC`(RBF Kernel)
  - Ensemble (`VotingClassifier`) of: `RandomForestClassifier`, `SVM.SVC`(RBF Kernel), and `CatboostClassifier`
- **Hyper-Parameter Tuning:** In some cases `RandomizedSearchCV` (vs. `GridSearchCV`) is preferred since it can search a continuous distribution (vs. only discrete lists in the `GridSearchCV` case - in order to overcome the potential flaw "did I not include an important possible value?")
- **Experimentation: ensembling can "squeeze out" even more accuracy**
- **Produces unlabeled predictions used in competition scoring**

Best Competition Model: Summary

- **Algorithms used:** Ensemble (`VotingClassifier`) of: `RandomForestClassifier` and `CatboostClassifier`
  - “soft” voting results in greater accuracy than “hard” voting
  - the model from this ensemble results in greater leaderboard accuracy than either classification algorithm by itself
- **Leaderboard Stats:**

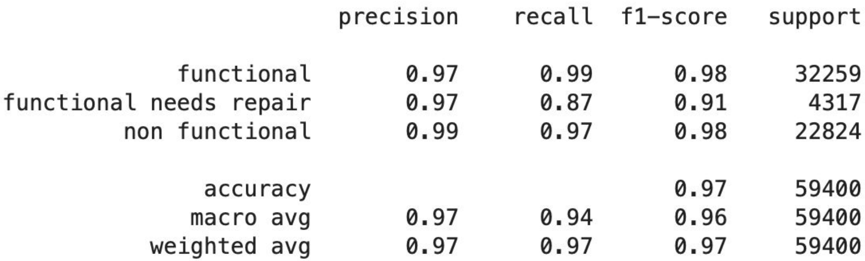


Submissions

BEST	CURRENT RANK	# COMPETITORS	SUBS. MADE
0.8187	837	9751	2 of 3

Best Competition Model: Summary: Part 2

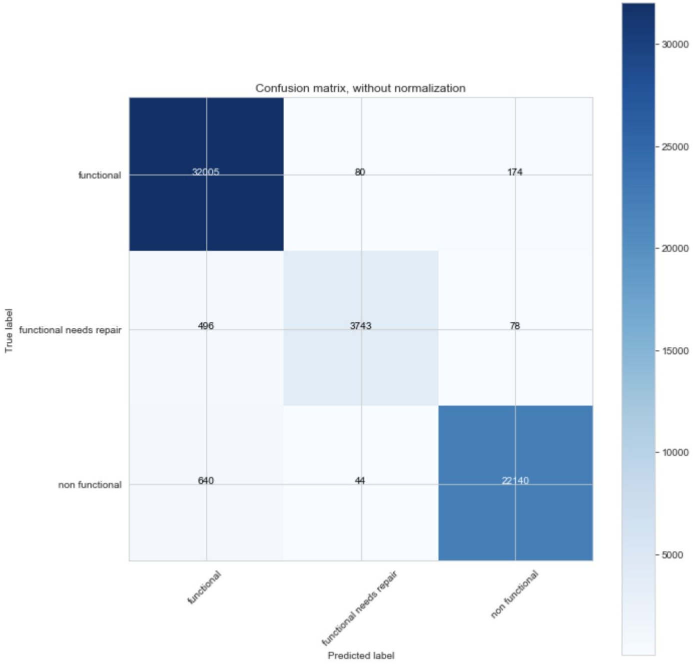
- **Accuracy:**
    - All-labeled (Train + Validation):   97.45%
- **Classification Report:**



```
                         precision    recall  f1-score   support

             functional       0.97      0.99      0.98     32259
functional needs repair       0.97      0.87      0.91      4317
         non functional       0.99      0.97      0.98     22824

               accuracy                           0.97     59400
              macro avg       0.97      0.94      0.96     59400
           weighted avg       0.97      0.97      0.97     59400
```



Best Competition Model: Feature Importances: 0

`VotingClassifier` **does not itself support feature importances, so we consider feature importances of each base classifier within the ensemble.**

Best Competition Model: Feature Importances: 1

`RandomForestClassifier` feature importances (ordered segment):

```
('longitude', 0.1419938300000846),
('latitude', 0.1408485505887231),
('ward_target_encoded', 0.13300125845632754),
('gps_height', 0.07072012309118425),
('construction_year', 0.05129212504533979),
('population', 0.04792955985377234),
('waterpoint_type_target_encoded', 0.04523917965056266),
('lga_target_encoded', 0.04392610310562060 4),
('quantity_enough', 0.03280674663306963),
('region_code_target_encoded', 0.026483118450123147),
('extraction_type_other', 0.02128081908344558 7),
('management_target_encoded', 0.021228146094434134),
('quantity_insufficient', 0.02076935581412387),
('amount_tsh', 0.01674554733198661),
('quantity_seasonal', 0.011250677364134225),
…
```

(from here, specific/individual one-hot encoded features determine importance)

Best Competition Model: Feature Importances: 2

`CatboostClassifier` **feature importances:**

```
('ward_target_encoded', 22.41649207493856),
('longitude', 7.689541298355761),
('latitude', 7.5612363143697205),
('waterpoint_type_target_encoded', 6.783655095530763),
('gps_height', 5.963876290835135),
('construction_year', 5.590341595095007),
('population', 4.475029132086664),
('quantity_enough', 4.085805641935429),
('quantity_seasonal', 4.032185134327176),
('lga_target_encoded', 3.9554054760800943),
('quantity_insufficient', 3.7031852130628846),
('region_code_target_encoded', 2.488859587057985),
('management_target_encoded', 2.196357543701259),
('amount_tsh', 1.8572530354537147),
…
```

(from here, specific/individual one-hot encoded features determine importance)

Feature Importances: High-level Analysis

Though the feature-importance coefficients of the `RandomForestClassifier` and `CatboostClassifier` components of the final ensemble model differ, the feature-importance orderings of each are still rather close.

The similarity of feature-importance orderings provides a high-level overview of “final” feature-importance:

- ward_target_encoded
- longitude
- latitude
- waterpoint_type_target_encoded
- gps_height
- construction_year
- population
- lga_target_encoded
- region_code_target_encoded
- management_target_encoded
- amount_tsh

It is also very interesting that both classifiers in the ensemble agree on particular (one-hot encoded) categories that are of great importance in classifying the target:

- 'quantity_enough'
- 'quantity_seasonal'
- 'quantity_insufficient'

Conclusion and Future Work Consideration

Thank you so much for your time!

I really enjoyed this project and I learned MANY new and powerful techniques.

Possible Future Work:

- **More advanced ensemble techniques:**
  - **Stacking**
  - **Blending**
- **Use a different classification algorithm (e.g. `CatboostClassifier`) for baselining within the preprocessing.ipynb notebook for comparison**
- **Experiment more with `RandomizedSearchCV` for better resolution of hyper-parameter tuning candidate values with, for example, `RandomForestClassifier` which may lead to even higher competition accuracy**