Forecasting Patient Enrolment for Clinical Trials

6th Team Project Sprint Review
July 3rd, 2020

01

Preprocessing

Different preprocessing techniques necessary for the model

02

Data Gathering

Adding site-level information to the data

03

Pipeline and Custom Transformers

Setting up the evaluation and optimization framework

04

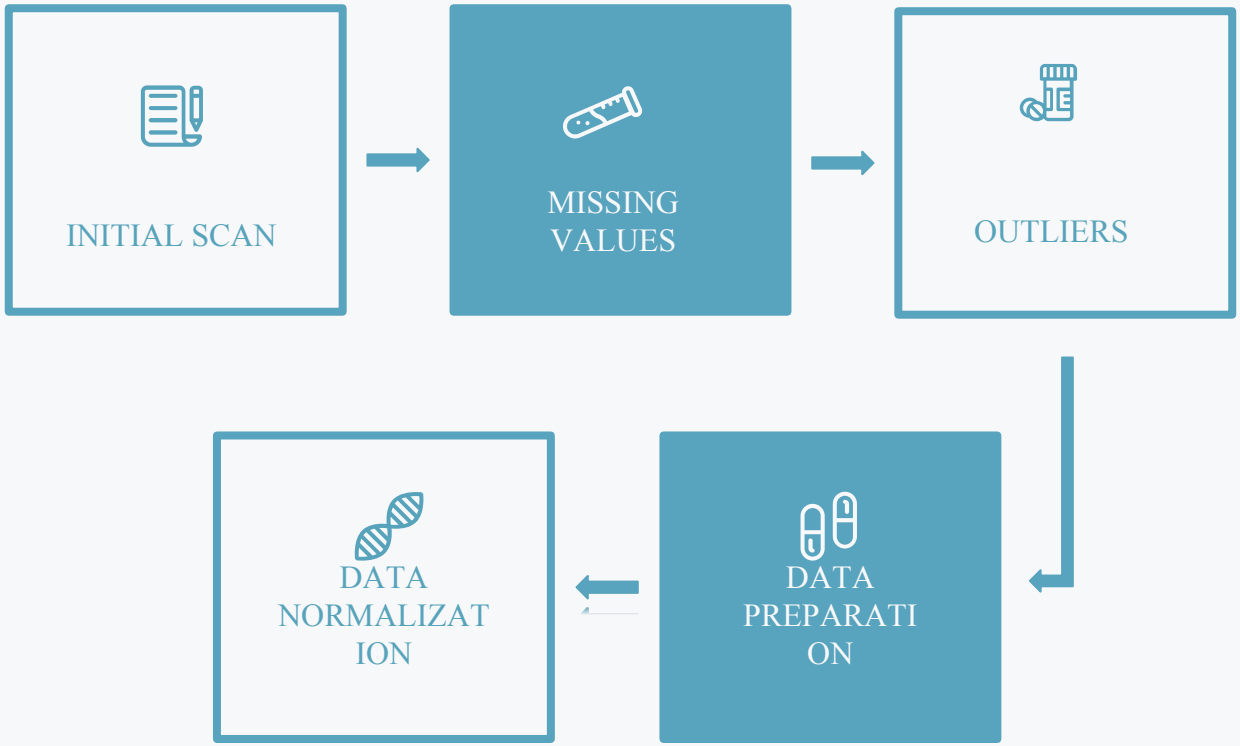
Hyperparameter Optimization

Optimizing the hyperparameters for all models

Preprocessing

- Preprocessing Approach
- Newly added features
- Using free text fields
- Using new condition related fields

Preprocessing Approach

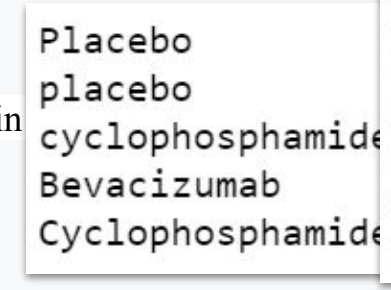


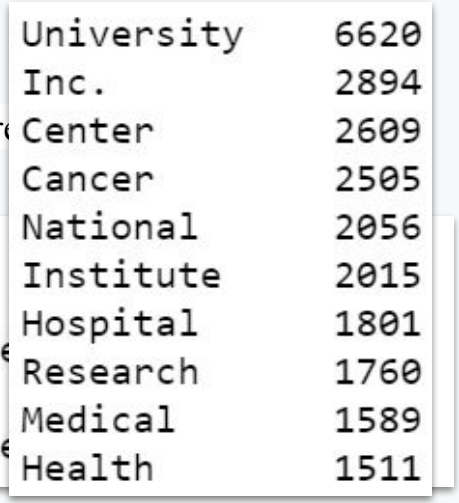
Newly added features

- Number of facilities, countries, ArmGroups, …
- Patients per site / country
- Average population, lifeExpectancy, …
- Extracted keywords
- Estimated enrollment count per country

Using free text fields

- Problem: Free text input fields of various length → One hot encoding leads to high dimensionality
- Examples:
  - InterventionName (i.e. [IncobotulinumtoxinA (16-20 Units per kg body weight)])
  - OrgFullName (i.e. Ankara City Hospital Bilkent)
- Approach:
  - Text processing (lower case, stopword removal, number & special character r
  - Tokenization of strings
  - Extract k - most frequent keywords
  - For each keyword do one hot encoding if it appears in original text field





Condition Branch

Conditions related to search results

See All Studies by Topic

Conditions
- Alphabetical
- By Category

Rare Diseases
- Alphabetical

Drug Interventions
- Alphabetical
- By Category

Dietary Supplements
- Alphabetical
- By Category

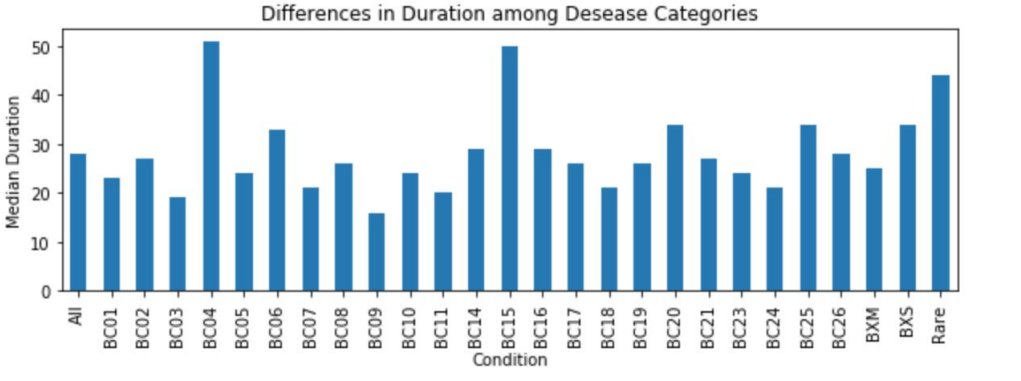
Sponsor/Collaborators
- Alphabetical
- By Category

Locations
- Alphabetical
- By Region

See Conditions by Category

- Bacterial and Fungal Diseases
- Behaviors and Mental Disorders
- Blood and Lymph Conditions
- Cancers and Other Neoplasms
- Digestive System Diseases
- Diseases and Abnormalities at or Before Birth
- Ear, Nose, and Throat Diseases
- Eye Diseases
- Gland and Hormone Related Diseases
- Heart and Blood Diseases
- Immune System Diseases
- Mouth and Tooth Diseases
- Muscle, Bone, and Cartilage Diseases
- Nervous System Diseases
- Nutritional and Metabolic Diseases
- Occupational Diseases
- Parasitic Diseases
- Respiratory Tract (Lung and Bronchial) Diseases
- Skin and Connective Tissue Diseases
- Substance Related Disorders
- Symptoms and General Pathology
- Urinary Tract, Sexual Organs, and Pregnancy Conditions
- Viral Diseases
- Wounds and Injuries

=Categories of Conditions

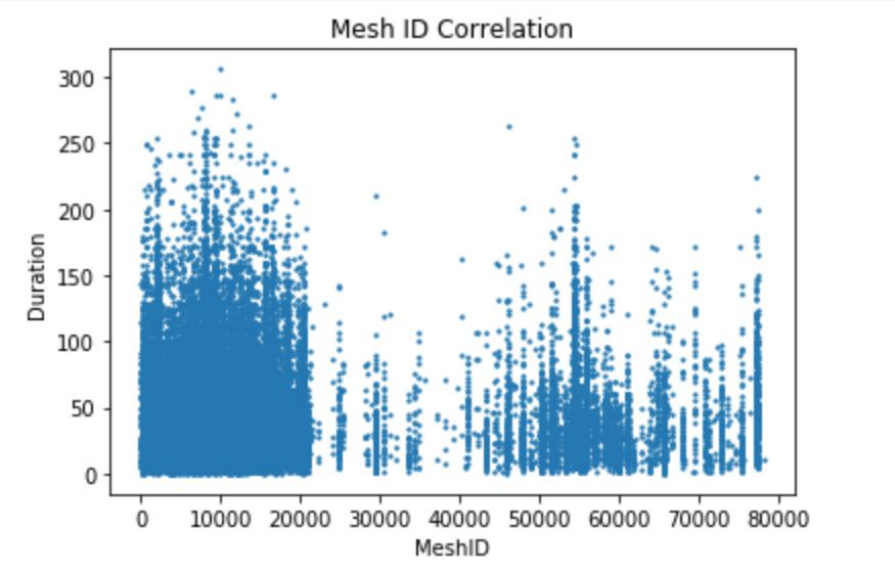


Condition MeshID

=Medical Subject Headings ID
= get additional Information on each condition
-> too complex

-IDs often have a naming convention / meaning
-use naive approach: treat ID as an integer

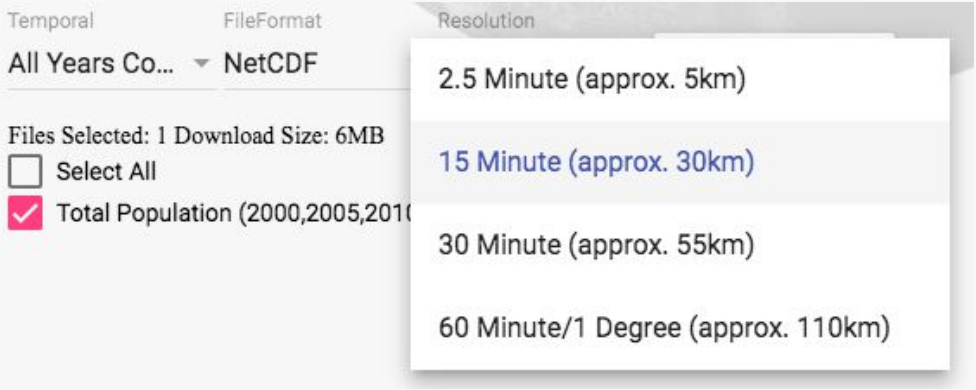
-mesh ID seems to be somehow correlated with duration



Data Gathering

- Local Population density
- additional regional-level information

Population density data



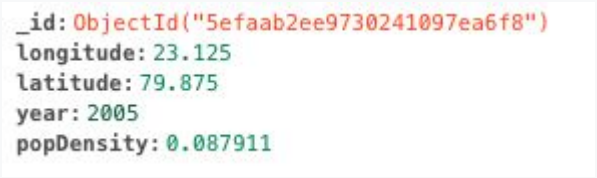
- Including data of 5 different years
- 4 different resolutions

=> Because of our db's space limitation, we use 15 minute-arc (~900km2 squares)

```
for i in nc.variables:
    print([i, nc.variables[i].units, nc.variables[i].shape])
```

```
['longitude', 'degrees_east', (1440,)]
['latitude', 'degrees_north', (720,)]
['raster', 'unknown', (20,)]
['Population Density, v4.11 (2000, 2005, 2010, 2015, 2020): 15 arc-minutes', 'Persons per square kilometer',
(20, 720, 1440)]
```

Population density data



Data extraction

Next steps:

- Get coordinates of location facilities by using Google Maps APIs
- Calculate population count on site-level based on k-nearest coordinates
- Distributing enrollment count based on the ratio of population count

Additional Regional Information

- If we cannot assign the longitude/latitude to a facility: use regional level
- if we cannot assign on the regional level: use country level

- More information might be inside SEDAC's data: Age and gender proportion
- More information to add on the regional level: Age Structure, Unemployment Rate and Wifi Access

Pipelines and Custom Transformers

- Using a pipeline and custom transformers
- Different approaches to build a pipeline
- Benefits of using a pipeline

Pipeline

- automates preprocessing and estimation steps
- ensures that all necessary steps are taken in the correct order
- connect preprocessing, hyperparameter optimization and model training in one function

Manage the complexity of our prediction model

1. Approach

I. Define Groups of similar features (in terms of transformation)

```
categorical_single_features = ['HealthyVolunteers','IsFDARegulatedDrug']
categorical_list_features = ['Phase', 'StdAge', 'CollaboratorClass']
numerical_features = ['EnrollmentCount']
```

I. Preprocessing: Apply a different pipeline on every group of features (including own defined transformation classes)

```
categorical_single_pipeline = Pipeline( steps = [
      ( 'cat_selector', FeatureSelector(categorical_single_features) ),
      ('cat_transformer', CategoricalEmptyValuesTransformer(strategy = "most_common") ),
      ( 'one_hot_encoder', OneHotEncoder() ) ] )
```

I. Estimation: Apply a pipeline on the different transformation pipelines and include the estimator

```
full_pipeline = Pipeline( transformer_list = [
      ( 'categorical_single_pipeline', categorical_single_pipeline ),
      ( 'categorical_list_pipeline', categorical_list_pipeline ),
      ( 'numerical_pipeline', numerical_pipeline ),
      ('random', RandomForestRegressor(p1, p2))] )
```

2. Approach

I. Define Groups of similar features (in terms of transformation)

```
Features that require one hot encoding (e.g. one_hot_features=[OrgClass,LeadsponsorClass,...])
Features that require binary encoding (e.g binary_features=[HealthyVolunteers, Gender,...])
Features that require special transformations
```

I. Import / Write Transformer for each group

```
from sklearn.preprocessing import OneHotEncoder, BinaryEncoder
class MeshIDTransformer(BaseEstimator, TransformerMixin):
      def __init__(self): ...
```

I. Combine all transformers as steps in a pipeline and pass them the groups they need to transform

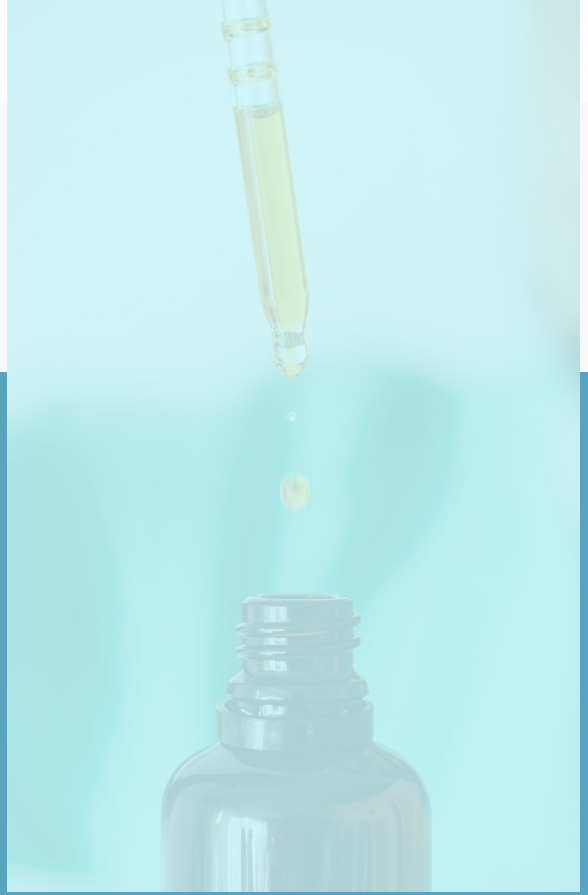
```
steps=[('one_hot_encoder', OneHotEncoder(one_hot_features)), ('binary_encoder',
BinaryEncoder(binary_features)),('meshID_transform', MeshIDTransformer("ConditionMeshID"))]

pipeline = Pipeline(steps)
```

I. Also add Feature Selection, Normalization, Outlier Detection, Hyperparameter Optimization and the actual Model into this pipeline

Benefits

- Organized readable code
- Definition of own Transformation classes allows reuse of code for similar features (DRY)
- Own transformation classes contain methods used by pipelines
  - inherited from TransformerMixin and BaseEstimator
  - *fit*, *transform* and *fit_transform* can be overwritten + additional methods such as *inverse_transform*
- Easily adjustable/reproducible, enables a simple change of the parameters
- Parameters used for Transformation are saved in the Transformation “instance” and can be applied on new data before prediction



Hyperparameter Optimization

- Set objective function to minimize
- Set spaces to search
- choose best overall model

1. Set Objective function

Define Objective function to minimize → Choose a loss function

Two possible settings (MAE or RMSE loss):

Cross Validation to find best loss

```
cv_results = lgb.cv(space_params, train, nfold = N_FOLDS, stratified=False,
                    early_stopping_rounds=100, metrics=EVAL_METRIC_LGBM_REG, seed=42)
best_loss = cv_results['l1-mean'][-1] #'l2-mean' for rmse
```

If we use some other algo (SVR, Random forest…) there is gridsearch CV available, different than lgb.cv.

1. Set Objective function

Define Objective function to minimize → Choose a loss function

Two possible settings + MAE or RMSE loss:

I. Cross Validation to find best loss

```
cv_results = lgb.cv(space_params, train, nfold = N_FOLDS, stratified=False,
                    early_stopping_rounds=100, metrics=EVAL_METRIC_LGBM_REG, seed=42)
best_loss = cv_results['l1-mean'][-1] #'l2-mean' for rmse
```

Problem encounter here:
If we use some other algo (SVR, Random forest…) there is only gridsearch CV available, different than lgb.cv.

I. No cross-validation but fit the model by setting the function below in the beginning

```
train_X, val_X, train_y, val_y = train_test_split(data,labels,test_size=0.2,random_state=1)
```

2. Hyperopt - Set Space

- Space over which to search can be like

hp.pchoice/choice	Choice with/without probability
hp.uniform(y, low, high)	Draws uniformly between low and high (continuous)
hp.quniform(y, low, high, q)	round(uniform(low, high) / q) * q (discrete)
hp.loguniform(y, low, high)	exp(uniform(low, high)) (normally for decimals)
….	…..

```
space ={'boosting' : hp.choice('boosting', boosting_list),
        'num_leaves' : hp.quniform('num_leaves', 2, LGBM_MAX_LEAVES, 1),
        'max_depth': hp.quniform('max_depth', 2, LGBM_MAX_DEPTH, 1),
        'max_bin': hp.quniform('max_bin', 32, 255, 1),
```

Reference paper

3. Run Model - Choose best overall model

- So far lightGBM/ XGBoost/ Random Forest …..
- Set objectives and spaces for each algorithm individually, because they have different function call
- Define another function to call the best
- With this function that returns the real number to compare loss for cv with lgb/ xgboost

```
trials.best trial['result']['loss']
```

- Find the model with least loss

There is 2 way possibly to be done by hyperopt, to choose the model with hp.pchoice, but it is somehow like

I. Use hp.choice to choose the model → use hyperopt to choose best parameter from the model with least loss
II. Use hyperopt estimator to use it in a pipeline way

Next Steps

Next Steps

- Apply the framework to all features
- How to combine Pipelines and Hyperopt?
- More MongoDB storage

Cluster Tier

M2 (Shared RAM, 2 GB Storage)
Encrypted

Base hourly rate is for a MongoDB replica set with **3 data bearing servers**.

Shared Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M0 Sandbox	Shared	512 MB	Shared	Free forever
M2	Shared	2 GB	Shared	$9 / MONTH
500 max connections | Low network performance | 100 max databases | 500 max collections				
M5	Shared	5 GB	Shared	$25 / MONTH