Bankruptcy Prediction

Team: **Data4Breakfast**

kaggle

Abhinav Chanda

Pranay Khandelwal

Sachin Arakeri

Project Overview

Objective	▪ **Develop a Predictive model** to assess whether a firm goes bankrupt or not
Dataset Provided	▪ Training Dataset (n=10,000 obs, 64 attributes) ▪ Test Dataset (n=5,000 obs, 64 attributes)
Platform	▪ Solution to be submitted on **Kaggle**
Project KPI	▪ AUC (Area under the curve) of ROC graph

Data Processing Techniques Applied

*Processing that did **NOT** yield satisfactory results*

- PCA
- Correlation
- Interaction Terms
- Log Transformation

Processing that were successful

- Drop Duplicates
- Oversampling
- Stratified K-Sample
- Hyper-parameter Tuning

Data Modeling Techniques Applied

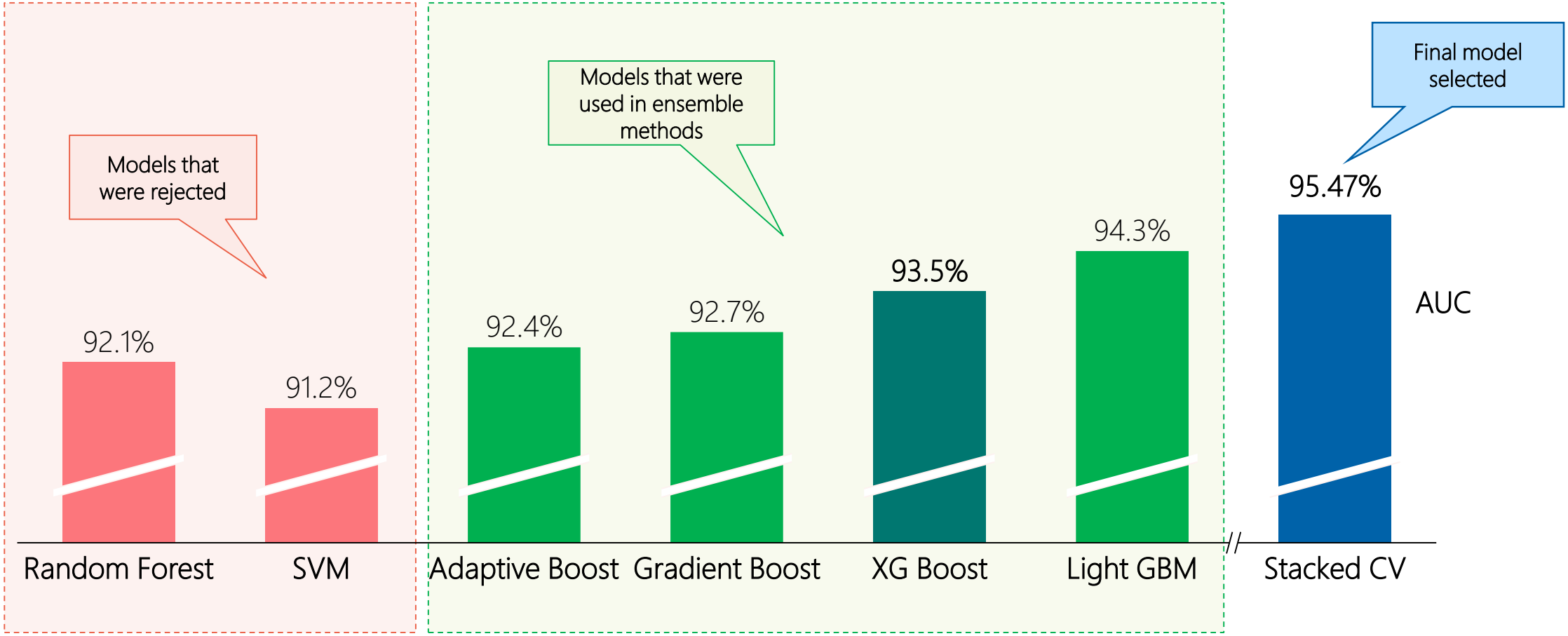
Models that were overfitting the data

- Gradient Boosting
- Random Forrest
- Adaptive Boosting
- Neural Networks

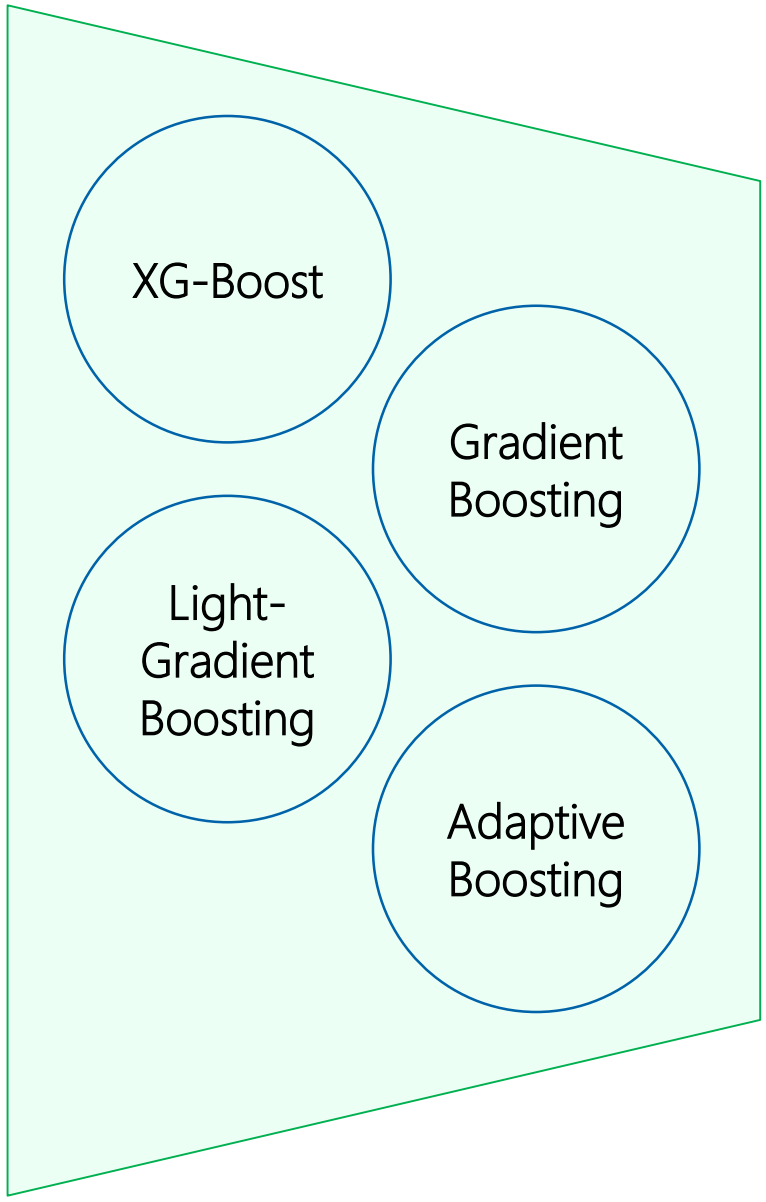
The models, when used in **isolation overfit the data** and led to sub par results

We decided to **apply ensemble technique** to overcome overfitting

Performance of models on validation dataset



Ensemble Method



XG Boost used at the Meta Classifier

Stacking CV Classifier

Ensemble: Used the four

Most Robust Model

96.25%
AUC (private)[1]

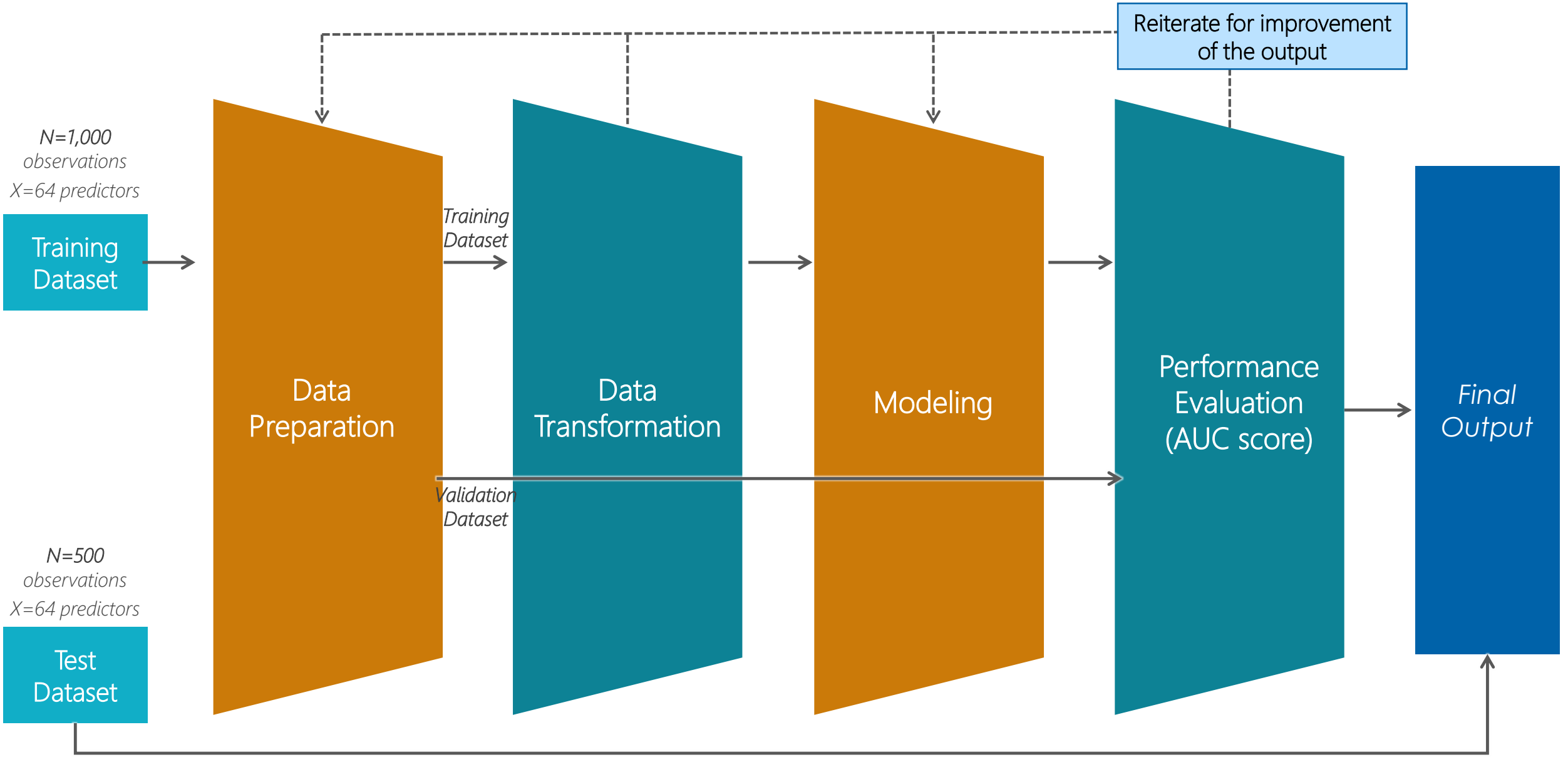
95.47%
AUC (public)[2]

We selected the model that had the least variability in results when applied on validation dataset and on the public test dataset

1. Based on the test dataset used for Kaggle's private leaderboard
2. Based on the test dataset used for Kaggle's public leaderboard

Thank You

Solution Process



Baseline Models

Run baseline XGBoost

In [5]:
```
xg_reg = XGBClassifier()
xg_reg.fit(x_train, y_train, eval_metric=eval_metric, eval_set=eval_set, verbose=False)
```

Out[5]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

Gradient Boost

In [6]:
```
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)
```

Out[6]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Hyperparameter Tuning

Hyper Parameter Tuning

In [9]:
```
#XGBoost
params = {"learning_rate": [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth": [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight": [ 1, 3, 5, 7 ],
 "gamma": [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree": [ 0.3, 0.4, 0.5 , 0.7 ] }
folds = 3
param_comb = 40
data_dmatrix = xgb.DMatrix(data=x_train,label=y_train)
skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)
random_search = RandomizedSearchCV(xg_reg, param_distributions=params, n_iter=param_comb, scoring='roc_auc',
                                   n_jobs=4, cv=skf.split(x_train,y_train), verbose=10, random_state=1001 )
random_search.fit(x_train, y_train)
print('\n Best estimator:')
print(random_search.best_estimator_)
```

```
Fitting 3 folds for each of 40 candidates, totalling 120 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:   18.0s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:   23.9s
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:   46.8s
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:   58.3s
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:  1.5min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:  1.9min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed:  2.2min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed:  2.6min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed:  3.1min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed:  3.6min
[Parallel(n_jobs=4)]: Done 105 tasks      | elapsed:  4.0min
[Parallel(n_jobs=4)]: Done 120 out of 120 | elapsed:  4.4min finished


 Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.0,
              learning_rate=0.2, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

Stacked Model

Stacked Model

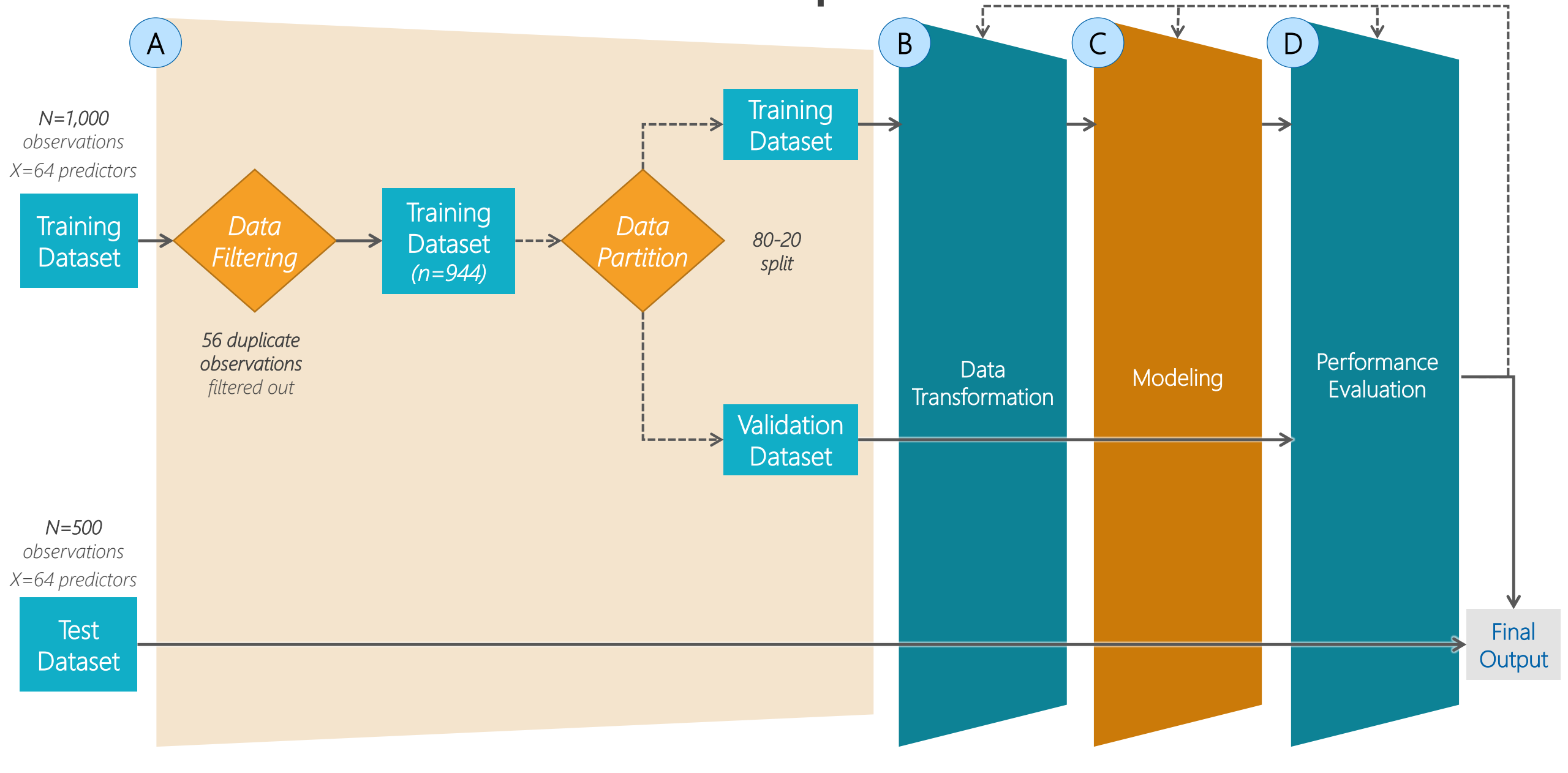
In [13]:
```
# Stack up all the models above, optimized using xgboost
stack_gen = StackingCVClassifier(classifiers=(xgb, abc, gbc, lgbm),
                                 meta_classifier=xgb,
                                 use_features_in_secondary=True)
stack_gen.fit(np.array(x_train), np.array(y_train))
```

Out[13]:
```
StackingCVClassifier(classifiers=(XGBClassifier(base_score=0.5,
                                                booster='gbtree',
                                                colsample_bylevel=1,
                                                colsample_bynode=1,
                                                colsample_bytree=0.4, gamma=0.0,
                                                learning_rate=0.1,
                                                max_delta_step=0, max_depth=3,
                                                min_child_weight=1,
                                                missing=None, n_estimators=300,
                                                n_jobs=1, nthread=None,
                                                objective='binary:logistic',
                                                random_state=0, reg_alpha=0,
                                                reg_lambda=1,
                                                scale_pos_weig...
                                                   n_estimators=300, n_jobs=1,
                                                   nthread=None,
                                                   objective='binary:logistic',
                                                   random_state=0, reg_alpha=0,
                                                   reg_lambda=1,
                                                   scale_pos_weight=1,
                                                   seed=None, silent=None,
                                                   subsample=1, verbosity=1),
                          n_jobs=None, pre_dispatch='2*n_jobs', random_state=None,
                          shuffle=True, store_train_meta_features=False,
                          stratify=True, use_clones=True,
                          use_features_in_secondary=True, use_probas=False,
                          verbose=0)
```

Tools and Libraries covered

S.N	Libraries used	Functions used	Application in our project
1	pandas	.read, .concat, .DataFrame,	data manipulation and cleaning
2	numpy	.power, .array, .linspace	Array manipulation
3	matplotlib	.plot, .show, .title	data visualization
4	sklearn	RandomForestClassifier, DecisionTreeClassifier, GradientBoostingClassifier, AdaBoostClassifier	Prediction models
5	scipy	stats	Random sample generation
6	xgboost	Fit, XGBClassifier, DMatrix,	Prediction model
7	mlxtend	StackingCVClassfier	Ensemble

Solution Framework:
A. Data Preparation



Solution Framework:
B. Data Transformation