```
from tqdm import tqdm

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

from sklearn.decomposition import PCA
from sklearn.decomposition import NMF
from sklearn.decomposition import FastICA
from sklearn.decomposition import TruncatedSVD

from sklearn.random_projection import SparseRandomProjection

from sklearn.feature_selection import f_regression
from sklearn.feature_selection import mutual_info_regression

from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score

import hypertools as hyp
import seaborn as sns
from seaborn import heatmap

%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot
from matplotlib.pyplot import scatter
```

```
y_train[:5]
train.head()
y_train.describe()
```

```
sns.set(style="white", palette="muted", color_codes=True)

# Set up the matplotlib figure
f, axes = plt.subplots(figsize=(13, 6), sharex=True)
sns.despine(left=True)

# Plot a simple histogram with binsize determined automatically
sns.distplot(y_train, hist=True, color="g", kde_kws={"shade": True}, ax
=axes)
plt.ylabel('Frequency')
plt.xlabel('Value')

#plt.setp(axes, yticks=[])
plt.tight_layout()
```

```
# Plotting Target Variable x = np.array(y_train) x.sort()
plt.figure(figsize=(17, 6)) plt.plot(x) plt.title('')
plt.ylabel('Value') plt.xlabel('Data points') plt.legend(['Target'],
loc='upper left') plt.show()
```

```
np.log1p(y_train).describe()
```

```
sns.set(style="white", palette="muted", color_codes=True)

# Set up the matplotlib figure
f, axes = plt.subplots(figsize=(13, 6), sharex=True)
sns.despine(left=True)

# Plot a simple histogram with binsize determined automatically
sns.distplot(np.log1p(y_train), hist=True, color="g", kde_kws={"shade":
True}, ax=axes)
plt.ylabel('Frequency')
plt.xlabel('Value')

#plt.setp(axes, yticks=[])
plt.tight_layout()
```

```
x = np.array(np.log1p(y_train))
x.sort()
plt.figure(figsize=(17, 6))
plt.plot(x)
plt.title('')
plt.ylabel('Value')
plt.xlabel('Data points')
plt.legend(['Target'], loc='upper left')
plt.show()
```

```
x = np.array(train[train.columns[0]]) plt.figure(figsize=(17, 6)) plt.plot(x)
plt.title('') plt.ylabel('Value') plt.xlabel('Data points') plt.legend(['Target'],
loc='upper left') plt.show()
```

```
sns.set(style="white", palette="muted", color_codes=True) # Set up the
matplotlib figure f, axes = plt.subplots(figsize=(13, 6), sharex=True)
sns.despine(left=True) # Plot a simple histogram with binsize determined
automatically sns.distplot(y_train, hist=True, color="g", kde_kws={"shade":
True}, ax=axes) plt.ylabel('Frequency') plt.xlabel('Value') #plt.setp(axes,
yticks=[]) plt.tight_layout()
```

```
x = np.array(train[train.columns[:7]])
plt.figure(figsize=(17, 6))
plt.plot(x)
plt.title('')
plt.ylabel('Value')
plt.xlabel('Data points')
plt.show()
```

```
x = np.log1p(np.array(train[train.columns[:7]])) plt.figure(figsize=(17, 6)) plt.plot(x)
plt.title('') plt.ylabel('Value') plt.xlabel('Data points') plt.show()
```

```
def check_sparsity(df): non_zeros = (df.ne(0).sum(axis=1)).sum() total =
train.shape[1]*train.shape[0] zeros = total - non_zeros sparsity =
round(zeros / total * 100,2) density = round(non_zeros / total * 100,2)
print(" Total:",total,"\n Zeros:", zeros, "\n Sparsity [%]: ",
```

```
sparsity, "\n Density [%]: ", density) return density d1 =
check_sparsity(train)
```

```
non_zeros_1 = (train.ne(0).sum(axis=1)) # number of zero elements colum
nwise
non_zeros_0 = (train.ne(0).sum(axis=0))

# Set up the matplotlib figure
f, axes = plt.subplots(figsize=(13, 6), sharex=True)
sns.despine(left=True)

# Plot a simple histogram with binsize determined automatically
sns.distplot(non_zeros_1, hist=True, color="g", kde_kws={"shade": True}
, ax=axes, bins = 100)
plt.ylabel('Frequency')
plt.xlabel('Value')
#plt.setp(axes, yticks=[])
plt.tight_layout()
```

```
                                                                    17]:
sns.set(style="white", palette="muted", color_codes=True)

# Set up the matplotlib figure
f, axes = plt.subplots(figsize=(13, 6), sharex=True)
sns.despine(left=True)
plt.ylabel('Frequency')
plt.xlabel('Value')
# Plot a simple histogram with binsize determined automatically
sns.distplot(non_zeros_0, hist=True, color="g", kde_kws={"shade": True}, ax
=axes, bins = 100)



plt.tight_layout()
```

```
# Set up the matplotlib figure
f, axes = plt.subplots(figsize=(18, 18), sharex=True)

# Plot a simple histogram with binsize determined automatically
heatmap(train, vmin=0, vmax=1, cmap="Blues")

plt.tight_layout()
pca = PCA(n_components=40, copy=True, whiten=False)
train_pca = pca.fit_transform(train)
plt.figure(figsize=(17, 6))
plt.ylabel('Component values')
plt.xlabel('Component values')



scatter(train_pca[:,0], train_pca[:,1], alpha = 0.1)
```

```
pca = PCA(n_components=40, copy=True, whiten=False) train_pca =
pca.fit_transform(np.log1p(train)) plt.figure(figsize=(17, 6)) plt.ylabel('Component
values') plt.xlabel('Component values') scatter(train_pca[:,0], train_pca[:,1],
alpha = 0.1)
```

```
x = np.array(corr) x.sort() plt.figure(figsize=(18, 6)) plt.plot(x, color = "b")
plt.title('') plt.ylabel('Mutual information') plt.xlabel('Features')
plt.legend(['Mutual information'], loc='upper left') plt.show()
```

```
x = np.array(corr[1]) x.sort() plt.figure(figsize=(18, 6)) plt.plot(x, color = "b")
plt.title('') plt.ylabel('p values') plt.xlabel('Features') plt.legend(['p values'],
loc='upper left') plt.show()
```

```
# Set up the matplotlib figure f, axes = plt.subplots(figsize=(18, 18),
sharex=True) # Plot a simple histogram with binsize determined automatically
heatmap(data_corr, vmin=0, vmax=1, cmap="Blues") plt.ylabel('Features')
plt.xlabel('Features') plt.tight_layout()
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
x_train = pca.fit_transform(encoded_train)
x_test = pca.transform(encoded_test)
explained_variance = pca.explained_variance_ratio_
explained_variance
```

```
xgb = xgboost.XGBRegressor(n_estimators=35, learning_rate=0.06, gamma=0,
subsample=0.6, colsample_bytree=0.7, min_child_weight=4, max_depth=3)
xgb.fit(x_train,y_train) predictions = xgb.predict(x_test)
print(metrics.mean_squared_error(y_test, predictions))
```

```
rand = RandomForestRegressor(n_estimators = 10,random_state = 0)
rand.fit(x_train,y_train)
y_pred2 = rand.predict(x_test)
print(metrics.mean_squared_error(y_test,y_pred2 ))

logreg=LinearRegression()
logreg.fit(x_train,y_train)
y_pred=logreg.predict(x_test)

y_pred
print(metrics.mean_squared_error(y_test, y_pred))
```

```
import os import gc import time import pickle import pandas as pd import numpy
as np import lightgbm as lgb from sklearn.metrics import
mean_squared_error
```

```
%%time
train_df_statistic = pd.read_csv(os.path.join(PATH_TO_DATA, 'train_with
_row_statistic_and_bin_thresh098.csv'))
test_df_statistic = pd.read_csv(os.path.join(PATH_TO_DATA, 'test_with_r
ow_statistic_and_bin_thresh098.csv'))

train_space_reduction = pd.read_csv(os.path.join(PATH_TO_DATA, 'train_s
pace_reduction_thresh098.csv'))
test_space_reduction = pd.read_csv(os.path.join(PATH_TO_DATA, 'test_spa
ce_reduction_thresh098.csv'))
%%time
```

```
train_df = pd.concat([train_df_statistic, train_space_reduction], axis=
1)
test_df = pd.concat([test_df_statistic, test_space_reduction], axis=1)

del train_df_statistic, test_df_statistic
del train_space_reduction, test_space_reduction
gc.collect()

print('Train:', train_df.shape)
print('Test:', test_df.shape)
```

```
br = BoostARoota(metric='rmse') br.fit(train_df, y)
```

```
print('Total features:', train_df.shape[1]) print('Number of selected
features:', len(br.keep_vars_.values))
```

```
train_df = pd.read_csv(os.path.join(PATH_TO_DATA, 'train_boruta_stat_bi
n_red_thresh098.csv'))
test_df = pd.read_csv(os.path.join(PATH_TO_DATA, 'test_boruta_stat_bin_
red_thresh098.csv'))
```

```
def get_20_cv_splits(data, in_path):
    #stratify_classes = y
    train = pd.read_csv(os.path.join(PATH_TO_DATA, 'input/train.csv'),
usecols=['target'])
    stratify_classes =  train.target.apply(lambda x: int(np.log10(x)))
    splits = {}
    for random_state in range(20):
        column = np.zeros(data.shape[0])
        sss = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_
state=random_state)
        for i, (_, test_index) in enumerate(sss.split(data, stratify_cl
asses)):
            column[test_index] = i

        splits["split{}".format(random_state)] = column

    pd.DataFrame(splits, index=data.index).to_csv(os.path.join(PATH_TO_
DATA, in_path))
```

```
def create_folds_from_cv_splits(in_path, pkl_path):

    cv_splits = pd.read_csv(os.path.join(PATH_TO_DATA, in_path))
    folds_list = []
    for ind, i in enumerate(cv_splits.columns[1:]):
        folds = list(set(cv_splits[i].values))
        folds_list.append([])
        for m in folds:
            val_idx = list(cv_splits[cv_splits[i]==m].index)
            train_idx = list(set(list(cv_splits.index)) - set(val_idx))
            folds_list[ind].append((train_idx, val_idx))
    with open(os.path.join(PATH_TO_DATA, pkl_path), 'wb') as f:
        pickle.dump(folds_list, f)
```

```
    return folds_list
```

```
LOAD_CV = True

if LOAD_CV:
    with open(os.path.join(PATH_TO_DATA, 'folds/custom_cv_boruta_thresh
098.pkl'), 'rb') as f:
        cv_folds = pickle.load(f)
else:
    get_20_cv_splits(train_df, in_path='folds/cv_splits_boruta_stat_bin
_red_thresh098.csv')
    cv_folds = create_folds_from_cv_splits(in_path='folds/cv_splits_bor
uta_stat_bin_red_thresh098.csv',
                                           pkl_path='folds/custom_cv_bo
ruta_thresh098.pkl')
```

```
def run_lgb(train_X, train_y, val_X, val_y, test_X):
    params = {
        "objective" : "regression",
        "metric" : "rmse",
        "num_leaves" : 40,
        'max_depth': 7,
        "learning_rate" : 0.005,
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.1,
        "bagging_frequency" : 6,
        "bagging_seed" : 44,
        "verbosity" : -1,
        'num_threads' : 4,
        "seed": 44
    }

    start_time = time.time()
    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    model = lgb.train(params, lgtrain, 5000,
                      valid_sets=[lgtrain, lgval],
                      early_stopping_rounds=100,
                      verbose_eval=150)
    print('Model training done in {} seconds.'.format(time.time() - sta
rt_time))

    pred_test_y = np.expm1(model.predict(test_X, num_iteration=model.be
st_iteration))
    pred_oof_log = model.predict(val_X, num_iteration=model.best_iterat
ion)
    return pred_test_y, pred_oof_log, model
```

```
def run_calculations(X, test, big_cv_folds, func_name = None): if not
func_name: return print('The function to run is not defined') else:
y_oof_20_preds = [] fold_errors_20_preds =[] avg_test_pred_20_preds =
[] fold_errors_std = [] for ind, cv_folds in enumerate(big_cv_folds):
print('Fitting big fold', ind+1, 'out of', len(big_cv_folds)) y_oof =
np.zeros((y.shape[0])) fold_errors =[] pred_test_list = [] for i,
```

```
(train_index, val_index) in enumerate(cv_folds): print('Fitting sub
fold', i+1, 'out of', len(cv_folds)) X_train, X_val =
X.iloc[train_index], X.iloc[val_index] y_train, y_val = y[train_index],
y[val_index] # part to include additional functions if func_name ==
'lgb': pred_test_y, pred_oof_log, clf = run_lgb(X_train, y_train,
X_val, y_val, test) elif func_name == 'xgb': pred_test_y, pred_oof_log,
clf = run_xgb(X_train, y_train, X_val, y_val, test) else: return
print('The function to run is not correct') y_oof[val_index] =
pred_oof_log curr_fe = np.sqrt(mean_squared_error(y_val, pred_oof_log))
print(f'Fold error {curr_fe}') fold_errors.append(curr_fe)
pred_test_list.append(list(pred_test_y)) print('Total error',
np.sqrt(mean_squared_error(y, y_oof))) total_fe_std =
round(np.std(fold_errors), 5) print(f'Total std {total_fe_std}')
avg_test_pred = np.mean(pred_test_list, axis=0)
avg_test_pred_20_preds.append(avg_test_pred)
fold_errors_20_preds.append(fold_errors) y_oof_20_preds.append(y_oof)
fold_errors_std.append(total_fe_std) return y_oof_20_preds,
avg_test_pred_20_preds, fold_errors_20_preds, fold_errors_std
```

```
%%time
y_oof_lgb, pred_test_list_lgb, fold_errors, fold_std = run_calculations
(train_df, test_df, cv_folds, 'lgb')
```

Loading reqired libraries

library('ggplot2')

library('scales')

library('grid')

library('gridExtra')

library('RColorBrewer')

library('corrplot')

library(dplyr)

df <- read.csv('train.csv')

dim(df) # 4459 rows , 4993 columns

str(df)

Cheking for sparsness

sum(df == 0)/(dim(df)[1]*dim(df)[2])

Analyzing our target column

```
p1 <- df %>%
  ggplot(aes(target)) +
  geom_histogram(bins = 100, fill = "red") +
  scale_x_log10() +
  labs(x = "Target") +
  ggtitle("target feature distribution")

p2 <- df %>%
  mutate(tar = as.character(target)) %>%
  group_by(tar) %>%
  count() %>%
  arrange(desc(n)) %>%
  head(10) %>%
  ggplot(aes(reorder(tar, n, FUN = min), n)) +
  geom_col(fill = "blue") +
  coord_flip() +
  labs(x = "Target values", y = "frequency") +
  ggtitle("Most frequent target values")

grid.arrange(p1, p2, layout_matrix = rbind(c(1,2)))
```

Analysisnf predictor variables

```
# Loading reqired libraries

library('data.table')
library('tibble')
```

```
library('tidyr')
library('stringr')
library('forcats')



var_means <- df %>%
  select(-ID, -target) %>%
  summarise_all(funs(mean)) %>%
  gather(everything(), key = "feature", value = "mean")

var_sd <- df %>%
  select(-ID, -target) %>%
  summarise_all(funs(sd)) %>%
  gather(everything(), key = "feature", value = "sd")



stat <- df %>%
  select(-ID, -target) %>%
  summarise_all(funs(sum(.<0.001))) %>%
  gather(everything(), key = "feature", value = "zeros") %>%
  left_join(var_means, by = "feature") %>%
  left_join(var_sd, by = "feature")

# Visualising the above

p1 <- stat %>%
  ggplot(aes(mean+1)) +
  geom_histogram(bins = 30, fill = "red") +
  scale_x_log10() +
  labs(x = "Feature mean + 1") +
  ggtitle("Feature means")
```

```
p2 <- stat %>%
  ggplot(aes(sd+1)) +
  geom_histogram(bins = 30, fill = "blue") +
  scale_x_log10() +
  labs(x = "Feature std dev + 1")  +
  ggtitle("Feature std dev")

p3 <- stat %>%
  mutate(zeros = zeros/nrow(df)*100) %>%
  ggplot(aes(zeros)) +
  geom_histogram(bins = 50, fill = "orange") +
  labs(x = "Percentage of zero values")  +
  ggtitle("Zero values in feature")

p4 <- stat %>%
  ggplot(aes(mean+1, sd + 1)) +
  geom_point(col = "darkgreen") +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Feature mean + 1", y = "Feature std dev + 1")

grid.arrange(p1, p2, p3, p4, layout_matrix = rbind(c(1,2),c(3,4)))



# Correlation with target variables
#spearman
spearman_correlations <- df %>%
  select(-ID, -target) %>%
  cor(df$target, method = "spearman") %>%
  as.tibble() %>%
```

```
  rename(cor = V1)


ggplot(spearman_correlations, aes(x=cor)) + geom_histogram() + labs(x = "Spearman Correlation coefficient") + ggtitle("Spearman Correlation of anonymous vs target")


# Pearson

pearson_correlations <- df %>%

  select(-ID, -target) %>%

  cor(df$target, method = "pearson") %>%

  as.tibble() %>%

  rename(cor_p = V1)


ggplot(pearson_correlations, aes(x=cor_p)) + geom_histogram() + labs(x = "Pearson Correlation coefficient") + ggtitle("Pearson Correlation of anonymous vs target")


### Building models

# removing ID column as is of no use

df$ID <- NULL


# Loading reqired libraries


install.packages('tidyverse')

install.packages('caret')

install.packages('glmnet')

library(tidyverse)

library(caret)

library(glmnet)

library(dplyr)

install.packages('psych')

library(psych)
```

```
attach(df)
y <- df %>% select(target) %>% scale(center = TRUE, scale = FALSE) %>% as.matrix()
x <- df %>% select(-target) %>% as.matrix()

dim(x)
dim(y)



### RIDGE Regression
# Perform 10-fold cross-validation to select lambda ---------------------------
lambdas_to_try <- 10^seq(-3, 5, length.out = 100)
# Setting alpha = 0 implements ridge regression
ridge_cv <- cv.glmnet(x, y, alpha = 0, lambda = lambdas_to_try,
                      standardize = TRUE, nfolds = 10)
# Plot cross-validation results
plot(ridge_cv)

# Best cross-validated lambda
lambda_cv <- ridge_cv$lambda.min
# Fit final model, get its sum of squared residuals and multiple R-squared
model_cv <- glmnet(x, y, alpha = 0, lambda = lambda_cv, standardize = TRUE)
y_hat_cv <- predict(model_cv, x)
ssr_cv <- t(y - y_hat_cv) %*% (y - y_hat_cv)
rsq_ridge_cv <- cor(y, y_hat_cv)^2
rsq_ridge_cv # 0.86

### LASSO Regression

# Perform 10-fold cross-validation to select lambda ---------------------------
lambdas_to_try <- 10^seq(-3, 5, length.out = 100)
# Setting alpha = 1 implements lasso regression
```

```
lasso_cv <- cv.glmnet(x, y, alpha = 1, lambda = lambdas_to_try,
                      standardize = TRUE, nfolds = 10)
# Plot cross-validation results
plot(lasso_cv)



# Best cross-validated lambda
lambda_cv <- lasso_cv$lambda.min
# Fit final model, get its sum of squared residuals and multiple R-squared
model_cv <- glmnet(x, y, alpha = 1, lambda = lambda_cv, standardize = TRUE)
y_hat_cv <- predict(model_cv, x)
ssr_cv <- t(y - y_hat_cv) %*% (y - y_hat_cv)
rsq_lasso_cv <- cor(y, y_hat_cv)^2 # 0.4968



#### Elastic Net Regression


library(caret)


# Set training control
train_control <- trainControl(method = "repeatedcv",
                              number = 5,
                              repeats = 5,
                              search = "random",
                              verboseIter = TRUE)


# Train the model
elastic_net_model <- train(target ~ .,
                           data = cbind(y, x),
                           method = "glmnet",
```

```
                    preProcess = c("center", "scale"),
                    tuneLength = 25,
                    trControl = train_control)

# Check multiple R-squared
y_hat_enet <- predict(elastic_net_model, x)
rsq_enet <- cor(y, y_hat_enet)^2 # 0.7119
```