Analysis of alloy steel composition-property relationship using machine learning

Raymond Wang[1]

[1] *Department of Materials Science and Engineering, Northwestern University, Evanston, Illinois 60208, USA*

This challenge project scrapes digital alloy steel data from online resources and converts them into machine-readable format. Data analysis using correlation heatmap shows degree of correlation within alloy steel composition and properties. Several strongly correlated quantities (e.g. hardness and tensile strength) are identified. Machine learning algorithm performance on predicting composition-property relationship is benchmarked using grid search. XGBoost outperforms the other methods in the alloy steel case. Further analysis shows predicting thermal conductivity from chemical composition using XGBoost has satisfying accuracy and the model performance can be improved by having more training data. Our findings suggest that machine learning methods could provide more insights of alloy steel composition-property relationship than using human physical intuition or experience.

I. WORKFLOW

The challenge project mainly consists of five sections, i.e. data acquisition, featurization, human interpretation, machine learning model training, and result analysis. The flowchart and specific outcomes in each step are shown in Figure 1. The rest of this report will span these five sections into details of implementation and obtained results. Data is available at the author's GitHub, the code can be found in supporting information (SI).

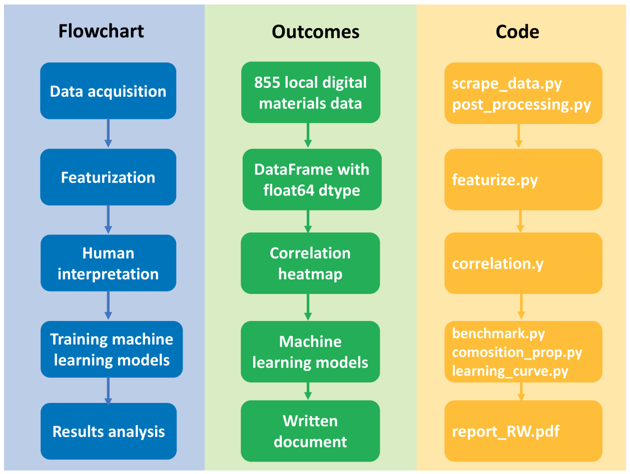

FIG. 1. Flowchart of the challenge project.

II. DATA ACQUISITION

The challenge project starts from acquiring materials data from online resources. Automated scraping code is developed by the author in Python, which automatically collects materials information that meets the searching criteria and saves to local CSV files.

Materials data is collected from MatWeb, where alloy steels containing Manganese, Chromium, and Nickel are set as the target materials for scraping. Each entry of material data spans four columns and multiple rows, as shown in Table I. Data under 'English' has different unit from 'Metric', this project only processes data with metric unit.

TABLE I. Format of online resources.

Property	Metric	English	Comments
Elongation at break	55%	55%	some comments
...	...	...	...

Physical properties (e.g. bulk modulus, thermal conductivity) in different units, chemical compositions (e.g. weight of Mn, Ni, Cr), and the potentially helpful comments, are saved to local files with unchanged format. Since there is no data-editing involved in this step, the local data is guaranteed to be the same as its online version.

One of the technical challenges during data acquisition is IP-blocking from the web server. MatWeb will block the IP (possibly permanently) once it detects over-access within a short period of time. From the author's experience, one IP address gives access to 100∼200 materials information per day. One of the advantage of the developed code is its high portability. The only two dependencies are `selenium` and `pandas` Python packages, which can be installed easily. In order to accelerate the project, the author used six machines with Unix-like systems for development and production. Eventually the program extracted 855 alloy steels that meet the criteria. The code (`scrape.py`) is available in SI.

Simple post-processing is performed right after all the data has been saved locally. This step modifies the file names and contents containing non-utf-8 encoding and fixes unwanted line breaks. Relevant code (`post_processing.py`) is available in SI.

III. FEATURIZATION

The data collected from the last step is still string-based, e.g. "97%" is interpreted as a sequence of characters instead of a floating point number. Therefore, it is necessary to convert these strings to machine-readable form before any further data analysis.

Since there are multiple materials properties and not all of them are available for each materials, dataset with missing values will be dropped. In order to keep a relatively large number of training set (e.g. several hundred), we only convert and keep physical properties with more than 100 available measured data points. These physical variables are: density, hardness (Vickers), thermal conductivity, specific heat capacity, CTE-linear, electrical resistivity, elongation at break, bulk modulus, modulus of elasticity, shear modulus, poisson's ratio, tensile strength at yield, and tensile strength at ultimate. Ten element types including Fe, Mn, Cr, Ni, Mo, Cu, C, S, Si, P are considered.

Floating point numbers are extracted from string data and converted to `pandas.DataFrame` format. The converted dataset is shown in Table II. Non-available data points are converted to `nan` instead of 'N/A'. The '%' and other units are dropped, only the floating point numbers are extracted. Since data will be standardized before numerical processing, the only thing to make sure is that all data in the same column share the same unit/percentage sign. The code (`featurize.py`) is available in SI.

TABLE II. Format of data after featurization.

Fe	Mn	Cr	···	Hardness	Bulk modulus	···
97.16	0.88	0.5	···	220	nan	···
95.43	0.85	0.0	···	nan	160	···
···	···	···	···	···	···	···

IV. DATA ANALYSIS WITH HUMAN INTELLIGENCE

The size of the dataset after featurization is 726×23 (with 726 instances and 23 features). It is impractical for humans (at least for the author) to directly learn patterns from such large amount of data. Based on basic statistical knowledge, the author decides to start from learning correlation patten of these variables. The instances with `nan` entries are dropped from the dataset, and eventually 254 materials are used to generate the heatmap, as shown in Figure 2. The code (`correlation.py`) can be found in SI.

Some of the discoveries from the heatmap include:

i. Carbon increases electrical resistivity but decreases shear modulus

ii. Sulfur increases specific heat capacity

iii. Phosphorus increases electrical resistivity

iv. Chromium decreases density

v. Molybdenum and copper strongly increases specific heat capacity and electrical resistivity

vi. Nickel increases thermal conductivity

vii. Hardness is positively correlated to tensile strength and negatively correlated to elongation at break

viii. Electrical resistivity is positively related to specific heat capacity

Some of the findings agree well with the way elements contribute to alloy steel properties as reported online, e.g. carbon decreases ductility of steel, and could lead to a small shear modulus. However, the information from correlation analysis is more qualitative than quantitative. If we want more quantitative descriptions of the materials composition-property relationship, more sophisticated methods are needed. In the next section we discuss alloy steel data analysis using machine learning models.

V. DATA ANALYSIS WITH ARTIFICIAL INTELLIGENCE

A. benchmark machine learning algorithm performance on dataset

The author performs systematic benchmark of different machine learning algorithm performance on various physical property predictions.

Each training takes one physical property as the target, while treats the rest 12 properties plus aforementioned 10 element types as input features. Dataset instances with `nan` are dropped from the dataset. Numerical data (both X and y) are standardized as implemented in `StandardScaler` in `sklearn`.

10 machine learning algorithms are benchmarked here, including:

1. Linear Regression
2. Least-angle regression with Lasso
3. Kernel Ridge
4. Linear SVR
5. SGD Regression
6. MLP Regressor
7. AdaBoost Regression
8. Random Forest Regression
9. Gradient Boosting Regression
10. Extremen Gradient Boosting

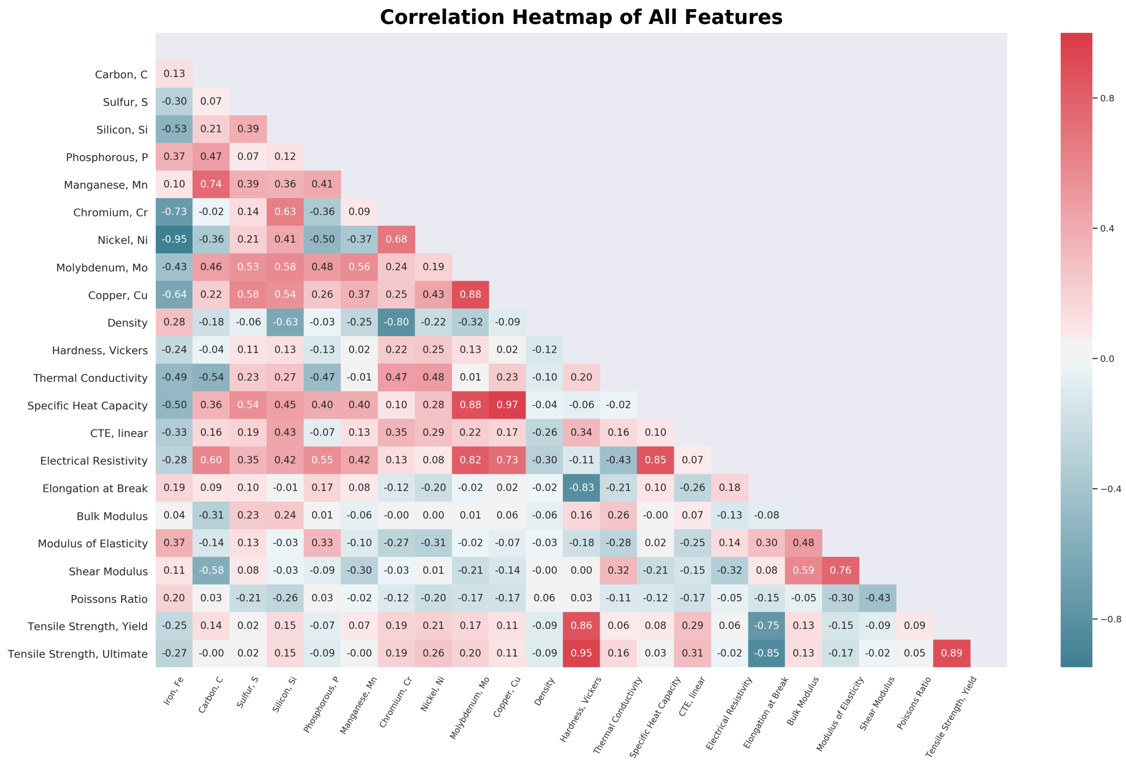

FIG. 2. Correlation heatmap of chemical composition and physical properties.

The dataset is divided into training (75%) and testing (25%) parts. Grid search method is used for hyper-parameter tuning. The set of hyper-parameters are shown in Table III . The optimal setting of the parameters is determined based on 5-fold cross-validation performed on training data only. R2 score is used as error metric. Relevant code (`benchmark.py`)can be found in SI. Results are shown in Figure 3.

The y-axis is the calculated root mean square error (RMSE) value divided by the difference of maximum and minimum value in the target data. This normalization step is essential so as to directly compare the performance of different machine learning algorithms in predicting quantities at various scales. The `sklearn` built-in performance scores (e.g. R2-score) are not used here.

Interestingly, some trends can be identified across different algorithms as well as physical properties. In general, linear regression, Lasso, linear SVR and SGD algorithms lead to larger variance in normalized RMSE, while the rest have relatively better performance. XGBoost algorithm achieves the best performance among the 10 benchmarked methods.

The physical properties are divided into two panels due to different scales of RMSE. We find it hard to have a good prediction in hardness and tensile strength, but it may not be a coincidence. It is exciting to notice that from the correlation heatmap analysis, it is clear that hardness is strongly and positively correlated to tensile strengths. This may suggest that these quantities are related to some other factors not considered here, such as processing and microscopic structures. And we need to include the other important features in order to build better machine learning models.

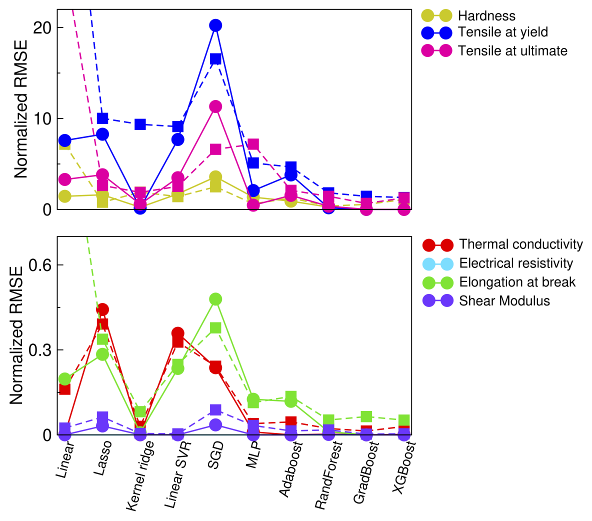

FIG. 3. Normalized root mean square error (RMSE) of different machine learning algorithms in predicting various physical properties. Training errors are shown in circles and solid lines, test errors are in squares and dashed lines.

B. predict composition-property relationship

We further investigate machine learning model performance on more challenging problems. In the previous section single property is targeted using both alloy chemical composition and other known physical properties. However in practical situations, sometimes other properties are also unknown (e.g. a new family of alloy steel), would machine learning be able to distill the composition-property relationship to facilitate alloy steel design?

We choose to use Extreme Gradient Boosting (XGBoost) algorithm for further analysis on alloy steel composition-relationship. This time the features only include composition information, i.e. weight percentage of each element. The performance of XGBoost in independently predicting 11 different physical properties is shown in Figure 4. R2-score is used as error metric in this case. The implementation of this part is very similar to that in `benchmark.py` with some small tweaks, the code is not attached in SI but can be found on the author's GitHub.

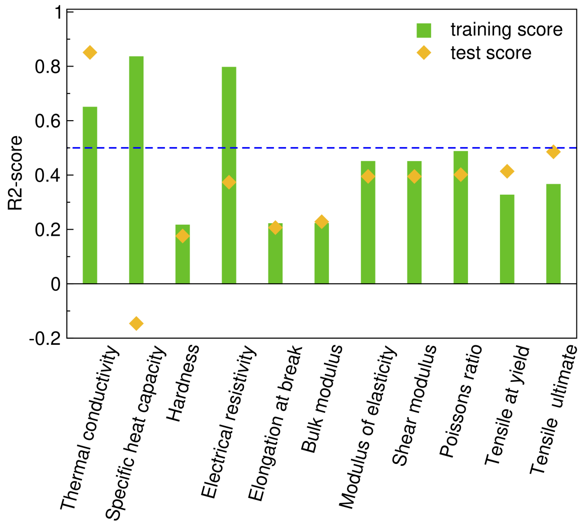

FIG. 4. R2-score of XGBoost in predicting 11 physical properties using only chemical composition as features. The dashed line is a border line above which has a 'satisfying' score.

Among the 11 tested properties, 8 of them do not show a strong relationship to composition, both training score and test score are below 0.5. Specific heat capacity and electrical resistivity get a high score in training but score poorly in test. This probably means XGBoost cannot correctly capture the relationship from the training set, but only numerically fit the data. In other words, the model suffers from overfitting. One thing to notice here is that R2-score can be negative, which means the model is even worse than a horizontal line through the mean feature value.

XGBoost performs well in predicting thermal conductivity using composition information, where the test score is even higher than training score. In order to validate our findings, the learning curve of predicting thermal conductivity is shown in Figure 5. As the number of training samples increase, cross-validation scores quickly increase and converge to the high training score. This clearly means adding more training samples can further improve the performance of XGBoost model in predicting thermal conductivity.

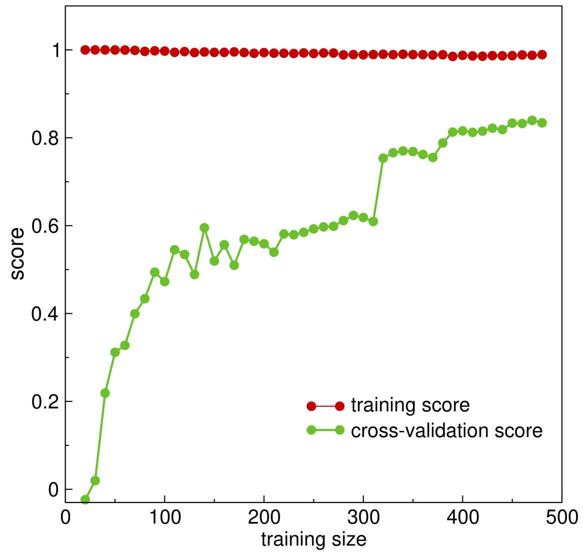

FIG. 5. Learning curve of predicting thermal conductivity from chemical composition using XGBoost algorithm.

VI. CONCLUSIONS

We find that machine learning could provide both qualitative and quantitative insights of alloy steel composition-property relationship. Gradient boost-based algorithms outperform the other machine learning algorithms in our benchmark. XGBoost is particularly successful in predicting thermal conductivity using alloy chemical composition, and the performance can be improved by adding more training samples.

TABLE III. Hyper-parameters used in different machine learning algorithms for grid search.

Algorithm	Parameter	Values
Linear Regression	default	default
Lasso	‘alpha’	{ 1E-4, 0.001, 0.01, 0.1, 1 }
Kernel Ridge	‘kernel’	{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’ }
	‘alpha’	{ 1E-4, 1E-2, 0.1, 1 }
	‘gamma’	{ 0.01, 0.1, 1, 10 }
Linear SVR	‘C’	{ 1E-6, 1E-4, 0.1, 1 }
	‘loss’	{ ‘epsilon_insensitive’, ‘squared_epsilon_insensitive’}
SGD	‘alpha’	{1E-6, 1E-4, 0.01, 1 }
	‘penalty’	{‘l2’, ‘l1’, ‘elasticnet’}
MLP	‘activation’	{‘logistic’, ‘tanh’, ‘relu’}
	‘solver’	{‘lbfgs’, ‘adam’, ‘sgd’}
	‘learning_rate’	{‘constant’, ‘invscaling’, ‘adaptive’}
Adaboost	‘n_estimators’	{10, 100, 1000}
	‘learning_rate’	{0.01, 0.1, 1, 10}
RandForest	‘n_estimators’	{10, 100, 1000}
	‘min_weight_fraction_leaf’	{0.0, 0.25, 0.5}
	‘max_features’	{‘sqrt’, ‘log2’, None}
GradBoost	‘n_estimators’	{10, 100, 1000}
	‘min_weight_fraction_leaf’	{0.0, 0.25, 0.5}
	‘max_feature’	{‘sqrt’, ‘log2’, None}
XGBoost	‘n_estimators’	{10, 50, 100, 250, 500, 1000}
	‘learning_rate’	{1E-4, 0.01, 0.05, 0.1, 0.2}
	‘gamma’	{0, 0.1, 0.2, 0.3, 0.4}
	‘max_depth’	{6}
	‘subsample’	{0.5, 0.75, 1}

VII. SUPPORTING INFORMATION

scrape.py

```
#! /home/raymond/.conda/envs/scrape/bin/python
# Author: Raymond Wang <raymondwang@u.northwestern.edu>
# Challenge project from Citrine Informatics
# Functionality: scrape alloy steel data from MatWeb and save to csv files

from selenium import webdriver
from selenium.webdriver.support.ui import Select
import pandas as pd
import numpy as np
import time
import sys
import os.path
import os

# navigate to target website
options = webdriver.ChromeOptions()
options.add_argument(”headless”) # comment this for visualization
driver_path = os.getcwd() + ”/chromedriver−linux” # google−chrome driver for Linux
#driver_path = os.getcwd() + ”/chromedriver−mac” # google−chrome driver for macOS
driver = webdriver.Chrome(executable_path=driver_path, chrome_options=options)
driver.get(”http://www.matweb.com/index.aspx”)

# user login, less likely to be blocked
driver.find_element_by_link_text(”LOG IN”).click()
time.sleep(2)
username = driver.find_element_by_id(”ctl00_ContentMain_txtUserID”)
username.send_keys(”USERNAME”)
passwd =  driver.find_element_by_id(”ctl00_ContentMain_txtPassword”)
passwd.send_keys(”PASSWD”)
time.sleep(0.5)
driver.find_element_by_xpath(”//input[@src='/images/buttons/btnLogin.gif']”).click()
time.sleep(3)

# Alloy composition
driver.find_element_by_link_text(”Alloy Composition”).click()

# unfold ”Ferrous Metal”
driver.find_element_by_id(”ctl00_ContentMain_ucMatGroupTree_LODCS1_msTreeViewn1”).click()
time.sleep(2)

# choose ”Alloy Steel”
driver.find_element_by_id(”ctl00_ContentMain_ucMatGroupTree_LODCS1_msTreeViewt6”).click()
time.sleep(2)

# choose composition1: Mn
select_fe = Select(driver.find_element_by_id(
    ”ctl00_ContentMain_ucPropertyDropdown1_drpPropertyList”))
select_fe.select_by_visible_text(”Manganese, Mn”)
driver.find_element_by_id(
    ”ctl00_ContentMain_ucPropertyEdit1_txtpMin”).send_keys(”0.0”)
time.sleep(2)
```

```

# choose composition2: Cr
select_cr = Select(driver.find_element_by_id(
    "ctl00_ContentMain_ucPropertyDropdown2_drpPropertyList"))
select_cr.select_by_visible_text("Chromium, Cr")
driver.find_element_by_id("ctl00_ContentMain_ucPropertyEdit2_txtpMin").send_keys("0.0")
time.sleep(2)

# choose composition3: Ni
select_ni = Select(driver.find_element_by_id(
    "ctl00_ContentMain_ucPropertyDropdown3_drpPropertyList"))
select_ni.select_by_visible_text("Nickel, Ni")
driver.find_element_by_id("ctl00_ContentMain_ucPropertyEdit3_txtpMin").send_keys("0.0")
time.sleep(2)

# lauch searching process
driver.find_element_by_id("ctl00_ContentMain_btnSubmit").click()
time.sleep(3)

# show 200 per page
instance_per_page = 200
Select(driver.find_element_by_id("ctl00_ContentMain_UcSearchResults1_drpPageSize1"))
    .select_by_visible_text(str(instance_per_page))
time.sleep(3)

# loop over 5 pages of results to collect data
npages = 5
for ipage in range(npages):
    print("currently on page " + str(ipage+1))
    sys.stdout.flush()

    # save name of datasets to a list before processing,
    # may not be ideal design but avoids stale element problems
    name_list = []
    for item in driver.find_elements_by_xpath(
        "//td[@style='width:auto; font-weight:bold;']"):
        name_list.append(item.text)

    for name in name_list:
        # preprocessing
        fname = name.replace(" ", "_")
        fname = fname.replace("/", "_")
        fname = fname.replace(",", "")
        # in case length of file name exceeds Linux limit (255B)
        if len(fname) > 240: fname = fname[:240]+"..."
        pathname = "data_raw/" + fname + ".csv"
        if os.path.isfile(pathname):
            if os.stat(pathname).st_size == 0: # in case last written unsuccessful
                os.remove(pathname)
            else:
                continue

        sys.stdout.flush()

        # open file
        f = open(”data_raw/” + fname + ”.csv”, ’w’)

        # navigate into each alloy page
        driver.find_element_by_link_text(name).click()
        time.sleep(5)
        table = driver.find_element_by_xpath(”//table[@class=’tabledataformat’]”)
        attrib = []
        for row in table.find_elements_by_xpath(”//tr[@class=’altrow datarowSeparator’]”):
            attrib.append([d.text for d in row.find_elements_by_css_selector(’td’)])
            time.sleep(0.5)
        for row in table.find_elements_by_xpath(”//tr[@class=’ datarowSeparator’]”):
            attrib.append([d.text for d in row.find_elements_by_css_selector(’td’)])
            time.sleep(0.5)
        attrib = np.array(attrib)

        # write to file
        df = pd.DataFrame(data=attrib[:, 1:], columns=[’Metric’, ’English’, ’Comments’],
            index=attrib[:, 0] )
        df.to_csv(f)
        f.close()
        driver.back()
        time.sleep(5)

    # navigate to the next page
    driver.find_element_by_id(”ctl00_ContentMain_UcSearchResults1_lnkNextPage”).click()
    time.sleep(5)

# quit chrome driver
driver.quit()
```

post_processing.py

```
import os

ddir = "./data_raw/"
# modify filenames
for filename in os.listdir(ddir):
    newname = filename
    for ch in filename:
        if (ch in ['.', '_', '-']) or (47 < ord(ch) 58) or (64 < ord(ch) < 91)
             or (96 < ord(ch) < 123): continue
        else: newname = newname.replace(ch, "")
    newname = newname.replace("__", "_")
    os.rename(ddir+filename, ddir+newname)

# fix unwanted new lines and non-utf-8 encodings
for filename in os.listdir(ddir):
    content = open(ddir+filename, errors='ignore').read()
    new_cont = content.replace("\n@", " @")
    f = open(ddir+filename, 'w')
    f.write(new_cont)
    f.close()
import os

# fix feature name problems
str1 = "string_to_be_replaced"
str2 = "new_string"
for filename in os.listdir(ddir):
    content = open(ddir+filename, errors='ignore').read()
    new_cont = content.replace(str1, str2)
    f = open(ddir+filename, 'w')
    f.write(new_cont)
    f.close()
```

featurize.py

```
from collections import defaultdict
import pandas as pd
import os

## select features and convert from strings to floating point numbers

features_dict = defaultdict(int)

ddir = "../scrape_data/data_raw/"
for filename in os.listdir(ddir):
    data = pd.read_csv(ddir+filename, header=0, index_col=0)
    for ind in data.index:
        features_dict[ind] += 1

print('These are the potential features:')
for ii in sorted(features_dict.items(), key=lambda tup: tup[-1], reverse=True):
    if (ii[-1] > 100) and (ii[0] is not ' '):
        print(ii)
print('')

elem_list = ['Iron, Fe', 'Carbon, C', 'Sulfur, S', 'Silicon, Si',
             'Phosphorous, P', 'Manganese, Mn', 'Chromium, Cr',
             'Nickel, Ni', 'Molybdenum, Mo', 'Copper, Cu']

prop_list = ['Density', 'Hardness, Vickers', 'Thermal Conductivity',
             'Specific Heat Capacity', 'CTE, linear', 'Electrical Resistivity',
             'Elongation at Break', 'Bulk Modulus', 'Modulus of Elasticity',
             'Shear Modulus', 'Poissons Ratio', 'Tensile Strength, Yield',
             'Tensile Strength, Ultimate']

feature_list = elem_list + prop_list
nfeature = len(feature_list)

# collect features from files
data_all_string = []
for filename in os.listdir(ddir):
    instance = ['0%']*len(elem_list) + ['NA']*len(prop_list) # initialize instance row
    data = pd.read_csv(ddir+filename, header=0, index_col=0)
    for ii in range(nfeature):
        if feature_list[ii] in data.index:
            instance[ii] = data.loc[feature_list[ii], 'Metric']
            # get rid of some unexpected '\n' in dataset
            if '\n' in instance[ii]: instance[ii] = instance[ii].replace('\n', ' ')
    if instance[0] is not 0: data_all_string.append(instance) # steel only

# convert to DataFrame and save to file
data_all_string = pd.DataFrame(data_all_string, index=None, columns=feature_list)
data_all_string.to_csv(path_or_buf="../data/data_all_string.csv", sep=';', index=False)

```

```
# define util function
def extract_float(string):
    if string is 'NA': return string # do not process non-available data
    num1 = ''
    num2 = ''
    curr = 0
    while not string[curr].isdigit(): # locate the first digit
        curr += 1
    # process the first float
    assert(string[curr].isdigit())
    while string[curr].isdigit():
        num1 += string[curr]
        curr += 1
        if curr == len(string): return float(num1)
    if string[curr] is '.':
        num1 += '.'
        curr += 1
        while string[curr].isdigit():
            num1 += string[curr]
            curr += 1
            if curr == len(string): return float(num1)
    # check if there is a second
    while string[curr].isspace():
        curr += 1
    if string[curr] is not '-':
        return float(num1)
    else: # process the second float
        curr += 1
    while string[curr].isspace():
        curr += 1
    assert(string[curr].isdigit())
    while string[curr].isdigit():
        num2 += string[curr]
        curr += 1
        if curr == len(string): return (float(num1) + float(num2)) / 2.0
    if string[curr] is '.':
        num2 += '.'
        curr += 1
        while string[curr].isdigit():
            num2 += string[curr]
            curr += 1
            if curr == len(string): return (float(num1) + float(num2)) / 2.0
    return (float(num1) + float(num2)) / 2.0
```

```python


# convert string to float
data_all_float = pd.DataFrame(data=data_all_string, copy=True)
for ii in range(data_all_float.shape[0]):
    for jj in range(data_all_float.shape[1]):
        data_all_float.iloc[ii, jj] = extract_float(data_all_float.iloc[ii, jj])

# drop instances whose Fe weight is less than 50%
drop_list = []
for ii in data_all_float.index:
    if data_all_float.loc[ii, 'Iron, Fe'] < 80.:
        drop_list.append(ii)
data_all_float = data_all_float.drop(drop_list)

print('Shape of data: ', data_all_float.shape)
data_all_float.to_csv("../data/data_all_float.csv", sep=';', index=False)
```

correlation.py

```
import pandas as pd
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
import numpy as np
import math
import os

data = pd.read_csv("../data/data_all_float.csv", header=0, index_col=None, sep=';')

# drop instances with NaN
drop_list = []
for idx in data.index:
    if True in [math.isnan(x) for x in data.loc[idx].values]:
        drop_list.append(idx)
data = data.drop(drop_list)
print('The shape of data: ', data.shape)

corr = data.corr()

fig = plt.figure(num=None, figsize=(40, 40), dpi=80, facecolor='w', edgecolor='w')
colormap = sns.diverging_palette(220, 10, as_cmap=True)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
ylabel = list(corr.columns)
ylabel[0] = ''
ax = sns.heatmap(corr, cmap=colormap, mask=mask, annot=True, fmt=".2f",
xticklabels=corr.columns[:-1], yticklabels=ylabel)
plt.title(label="Correlation Heatmap of All Features", loc='center', fontdict={
          'fontname':'DejaVu Sans', 'size':'24', 'color':'black', 'weight':'bold',
          'verticalalignment':'bottom'})
ax.tick_params(axis='x', rotation=60, labelsize=10)
ax.tick_params(axis='y', labelsize=13)
plt.show()
fig.savefig("heatmap_mask.png", dpi=200)
```

benchmark.py

```
import numpy as np
import pandas as pd
import sys
import operator
import math

import xgboost
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import *

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)


data = pd.read_csv("../data/data_all_float.csv", header=0, index_col=None, sep=';')

# drop 'CTE, linear' for more instances to study
drop_feature = ['Density', 'CTE, linear']
for ifeature in drop_feature:
    data = data.drop(labels=ifeature, axis=1)

# drop instances with NaN
drop_instance = []
for idx in data.index:
    if True in [math.isnan(x) for x in data.loc[idx, data.columns.values]]:
        drop_instance.append(idx)
data = data.drop(drop_instance)

print('Shape of dataset:')
print(data.shape)

hyper_params = [{
    'n_estimators' : (10, 50, 100, 250, 500, 1000,),
    'learning_rate' : (0.0001,0.01, 0.05, 0.1, 0.2,),
    'gamma' : (0,0.1,0.2,0.3,0.4,),
    'max_depth' : (6,),
    'subsample' : (0.5, 0.75, 1,),
}]

stdscale = StandardScaler()

properties = [ 'Thermal Conductivity', 'Specific Heat Capacity', 'Hardness, Vickers',
               'Electrical Resistivity', 'Elongation at Break', 'Bulk Modulus',
               'Modulus of Elasticity', 'Shear Modulus', 'Poissons Ratio',
               'Tensile Strength, Yield', 'Tensile Strength, Ultimate']

```

```
for target in properties:
    print('current target: ', target)

    X = data.drop(target, axis=1).values
    y = data[target].values

    # standardize features
    X = stdscale.fit_transform(X)
    y = stdscale.fit_transform(y.reshape(-1, 1))

    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75,
             test_size=0.25, random_state=0)

    regressor = xgboost.XGBRegressor()

    grid_clf = GridSearchCV(regressor, cv=5, param_grid=hyper_params,
             verbose=1, n_jobs=8, scoring='r2')

    grid_clf.fit(X_train, y_train.ravel())

    train_score_mse = mean_squared_error(stdscale.inverse_transform(y_train),
             stdscale.inverse_transform(grid_clf.predict(X_train)))
    test_score_mse = mean_squared_error(stdscale.inverse_transform(y_test),
             stdscale.inverse_transform(grid_clf.predict(X_test)))

    sorted_grid_params = sorted(grid_clf.best_params_.items(), key=operator.itemgetter(0)

    # print results
    out_txt = '\t'.join(['algorithm: ', str(sorted_grid_params).replace('\n', ','),
               str(train_score_mse), str(test_score_mse)])

    print(out_txt)
    print("")
    sys.stdout.flush()
```