EE460J - Lab 3

Can Gokalp (EID: CG39283), Priyadarshan Patil (EID: PP22352)

September 24, 2020

1 All questions

1.1 Problem 1

Read Shannon's 1948 paper 'A Mathematical Theory of Communication'. Focus on pages 1-19 (up to Part II), the remaining part is more relevant for communication. http://math.harvard.edu/ ctm/home/text/others/shannon/entropy/entropy.pdf Summarize what you learned briefly (e.g. half a page).

1.1.1 Solution to problem 1

Shannon's paper, "A Mathematical Theory of Communication", aims to tackle a fundamental problem of communication (as of 1948) relating to noiseless communication systems. The first part of the paper provides a brief intuition for the choice of the logarithmic function for information transfer, and then defines a communication system in terms of five components. The first two components are information source which provides the information to be transmit, and transmitter, which encodes the message in the form of a signal that can be transmit. The last two components are receiver and destination, which are the inverse of the transmitter and source, respectively. The transmitter and receiver are connected by a channel which is the medium used to transmit the signal, and that is where noise is likely to be introduced.

The next few subsections look at the mathematical properties of discrete noiseless systems. Starting with the capacity of a channel, there is a brief discussion on allowable sequences, sources of information, approximations and n-grams. Generally, these sections try to lay the foundation for modern natural language processing (NLP), by talking about series of approximations to English, establishing sentences as a Markov (Markoff?) process and specifically, Ergodic processes.

After this formulation, a measure of uncertainity is introduced. This measure is entropy, and it has to follow three properties relating to the probability distribution, i.e., continuity, monotonically increasing function of number of choices and indifference to successive choices. The only function satisfying said properties is the proposed "Shannon" formula which defines entropy as:

$$H = -K\Sigma_{i=1}^{n} p_i log p_i$$

A few properties of this formula are explored such as behavior at extremes, behavior under joint distributions, conditional entropy, etc. The last few subsections talk about application of entropy to an information source, and how encoding/decoding operations can be represented to minimize the number of required bits. The fundamental theorem for a noiseless channel provides a hard upper

bound for the average symbols per second transmitted for a given channel. Lastly, an example is provided to show how the average number of bits is obtained for a toy example with a special encoding scheme.

1.2 Problem 2

Scraping, Entropy and ICML papers

ICML is a top research conference in Machine learning. Scrape all the pdfs of all ICML 2017 papers from http://proceedings.mlr.press/v70/.

1. What are the top 10 common words in the ICML papers?
2. Let Z be a randomly selected word in a randomly selected ICML paper. Estimate the entropy of Z.
3. Synthesize a random paragraph using the marginal distribution over words.
4. (Extra credit) Synthesize a random paragraph using an n-gram model on words. Synthesize a random paragraph using any model you want. Top five synthesized text paragraphs win bonus (+30 points).

1.2.1 Solution to problem 2

The top 10 most common words and their frequencies can be found in Table 1 below.

Word	Count
the	150,968
of	76,160
and	66,081
in	54,055
to	50,705
cid	46,285
is	40,360
for	36,792
we	34,037
that	24,842

Table 1: Top 10 common words in ICML 2017

If **Z** is a randomly chosen word in a randomly selected ICML paper, the Shannon entropy of Z is calculated to be 10.855778128911782.

The paragraph below is a randomly generated paragraph from the marginal distribution over words (100 i.i.d draws):

> '13' 'the' 'now' 'optimal' 'xi' 'time' 'that' 'cases' 'algorithm' 'learning' 'quality' 'define' 'to' 'single' 'for' 'stein' 'sort' 'better' 'hs' 'of' 'variables' 'of' 'conference' 'and' 'to' 'group' 'on' 'and' 'on' 'problem' 'most' 'we' 'speech' 'processing' 'variables' 'machine' 'competitive' '21' 'is' 'supported' 'jarvis' 'low' 'unrolled' 'number'

'present' 'approach' '2017' 'christopher' 'assume' 'jumping' 'empirical' 'of' 'network' 'strict' 'error' 'the' 'all' 'of' 'seas' 'gradi' 'fact' 'v1' 'is' 'it' 'journal' 'more' 'the' 'prunes' 'policy' 'graph' 'using' 'are' 'machine' 'aims' 'know' 'than' 'the' 'little' 'learning' 'dual' 'the' 'references' 'machine' 'complexity' 'performance' '10' 'li' 'ciple' 'in' 'set' 'the' 'exist' 'range' 'with' 'some' 'achieves' 'private' 'is' 'experimentally'

The paragraph below is a randomly generated paragraph from the n-gram model (n=5):

The main motivation for considered the training cost can be reduced to the optimization problem, so running the model in real time may be learned from data, as well as when α grows sub-linearly to n. And$O(n^2)$ space embeddings of entities and relations can be the same function, we validate our theoretical results about 'mean predictor' and 'weisfeiler-lehman kernel with base kernel' at https://github.com/andersbll/theano since we would like to thank Jeff Johnson for his help but can potentially re-weight the units in the layer directly below, but scientifically interesting associations between discrepancy between barycenters and time series prediction length principle.

1.3 Problem 3

Starting in Kaggle

Soon you will be participating in the in-class Kaggle competition made for this class. In that one, you will be participating on your own. This is a warm-up - the more effort and research you put into this assignment the easier it will be to compete into the real Kaggle competition that you will need to do soon. We expect you to spend 10 times more effort on this problem compared to the others.

1. Let's start with our first Kaggle submission in a playground regression competition. Make an account to Kaggle and find https://www.kaggle.com/c/house-prices-advanced-regression-techniques/

2. Follow the data preprocessing steps from https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models. Then run a ridge regression using $\alpha = 0.1$. Make a submission of this prediction, what is the RMSE you get? (Hint: remember to exponentiate np.expm1(ypred) your predictions).

3. Compare a ridge regression and a lasso regression model. Optimize the alphas using cross validation. What is the best score you can get from a single ridge regression model and from a single lasso model?

4. Plot the l_0 norm (number of nonzeros) of the coefficients that lasso produces as you vary the strength of regularization parameter alpha.

5. Add the outputs of your models as features and train a ridge regression on all the features plus the model outputs (This is called Ensembling and Stacking). Be careful not to overfit. What score can you get? (We will be discussing ensembling more, later in the class, but you can start playing with it now).

6. Install XGBoost (Gradient Boosting) and train a gradient boosting regression. What score can you get just from a single XGB? (you will need to optimize over its parameters). We will discuss boosting and gradient boosting in more detail later. XGB is a great friend to all good Kagglers!

7. Do your best to get the more accurate model. Try feature engineering and stacking many models. You are allowed to use any public tool in python. No non-python tools allowed.

8. (Optional) Read the Kaggle forums, tutorials and Kernels in this competition. This is an excellent way to learn. Include in your report if you find something in the forums you like, or if you made your own post or code post, especially if other Kagglers liked or used it afterwards.

9. Be sure to read and learn the rules of Kaggle! No sharing of code or data outside the Kaggle forums. Every student should have their own individual Kaggle account and teams can be formed in the Kaggle submissions with partners. This is more important for live competitions of course.

10. As in the real in-class Kaggle competition (which will be next), you will be graded based on your public score (include that in your report) and also on the creativity of your solution. In your report (that you will submit as a pdf file), explain what worked and what did not work. Many creative things will not work, but you will get partial credit for developing them. We will invite teams with interesting solutions to present them in class.

1.3.1 Solution to problem 3

A Kaggle acocunt was made by my labmate and I. Our team is named SpartaLab. Following the data preprocessing steps from the provided link, then running ridge regression with $\alpha = 0.1$, we obtained a kaggle score of **0.13636**. Next, we tuned ridge and lasso regression alpha values. For tuned alpha values, we obtained best scores of: $ridge_{rmse}$: **0.12636** and $lasso_{rmse}$: **0.1232**. The l_0 norm (number of non-zero coefficients can be seen in Figure 1.

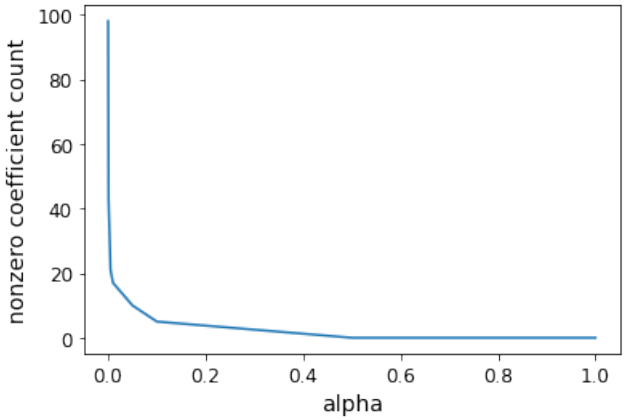

Figure 1: Variation of l_0 norm with alpha

The Kaggle score from the described stacking method in the problem was **0.13344**. This is better than the ridge regression score obtained earlier. One gradient boosted regression provided

mean rmse of **0.12709** and a Kaggle score of 0.132.

For part 7, we tried a number of things. We looked at the correlation matrix for features and eliminated some highly correlated features (e.g. GarageArea and GarageCars were highly correlated). Many variable pairs were visualized and a few outlier observations were removed. An example can be seen in Figure 2. Certain categorical features were dropped, such as Utilities and Street. They provided no additional information or predictive power.

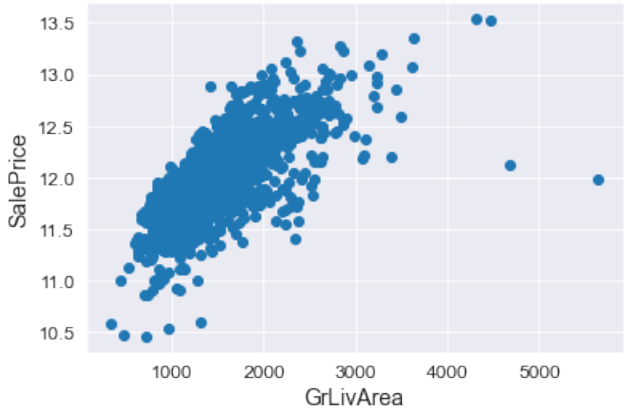

Figure 2: Feature visualization for outlier detection

After these steps, the remaining features were then analyzed for missing values. According to the data legend, "NA" values for certain columns meant that the feature was not present in the home, and not that the data was not available. For example, this can be seen for Garage related features, where NA means no garage. More detail can be found in the python notebook. All features with skewness above a certain threshold (0.7) were transformed to the logarithm of the feature to bring uniformity to the scale. Additional feature engineering methods involved combining features like various porch related areas into one metric, or calculating year of construction or remodelling.

After all these data processing steps, we fit the various regression models and optimized over available parameters. The details for each model and the score can be found in the notebook, but the best Kaggle score obtained was: **0.1239**. Overall, we learnt a lot in this assignment. However, it's hard to express that in the report except stating final scores. The notebook has much more detail and goes over everything we did.

Appendix A: Code for all problems

Importing required libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
```

```
mpl.rc('ytick', labelsize=12)
import pdfminer
import glob
import scipy
import math
import random
import sklearn
import os


import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr

from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from io import StringIO
from sklearn.feature_extraction.text import CountVectorizer
from nltk.lm import MLE
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.tokenize.treebank import TreebankWordDetokenizer

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import StackingRegressor
import xgboost

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR

from sklearn.compose import ColumnTransformer
from sklearn.feature_selection import VarianceThreshold

from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.linear_model import RidgeCV
#This block is just for importing relevant libraries
```

Code for problem 2

```
class PdfConverter:

    def __init__(self, file_path):
        self.file_path = file_path
# convert pdf file to a string which has space among words
    def convert_pdf_to_txt(self):
        rsrcmgr = PDFResourceManager()
        retstr = StringIO()
        codec = 'utf-8'  # 'utf16','utf-8'
        laparams = LAParams()
        device = TextConverter(rsrcmgr, retstr, codec=codec,
        laparams=laparams)
        fp = open(self.file_path, 'rb')
        interpreter = PDFPageInterpreter(rsrcmgr, device)
        password = ""
        maxpages = 0
        caching = True
        pagenos = set()
        for page in PDFPage.get_pages(fp, pagenos, maxpages=maxpages,
        password=password, caching=caching, check_extractable=True):
            interpreter.process_page(page)
        fp.close()
        device.close()
        str = retstr.getvalue()
        retstr.close()
        return str
# convert pdf file text to string and save as a text_pdf.txt file
    def save_convert_pdf_to_txt(self):
        content = self.convert_pdf_to_txt()
        txt_pdf = open('text_pdf.txt', 'ab')
        txt_pdf.write(content.encode('utf-8'))
        txt_pdf.close()

pdflist = glob.glob(r"C:\Users\priya\Dropbox\Sorted\UT Austin Academics
\Fall 20\EE 460 Data science lab\Lab 3\ICML Papers\*.pdf")

for pdf in pdflist:
    print("Working on: " + pdf + '\n')
    pdfConverter = PdfConverter(file_path=pdf)
    #print(pdfConverter.convert_pdf_to_txt())
    pdfConverter.save_convert_pdf_to_txt()

# Read all the stored plain text from pdfs
```

```
with open ("text_pdf.txt", "r", encoding="utf-8") as myfile:
    text=myfile.readlines()

# create the transform
vectorizer = CountVectorizer()
# Tokenize and build vocabulary from the text corpus
vectorizer.fit(text)
# Summarize the vocabulary
data = vectorizer.vocabulary_

# This writes the words and counts to a txt file
with open('Word_counts.txt', 'w', encoding = "utf-8") as f:
    print(vectorizer.vocabulary_, file=f)

# A function is defined to return top n frequent words from
a vocabulary list
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
    vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
    reverse=True)
    return words_freq[:n]

get_top_n_words(text, 10)

# This converts the counts to raw probabilities of appearance,
and drops zero value words, i.e., so rare that their probability
was rounded down to zero.
vec = CountVectorizer().fit(text)
bag_of_words = vec.transform(text)
sum_words = bag_of_words.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
print(type(words_freq))
Prob_dist_dict = dict()
def Convert(tup, di):
    for a, b in tup:
        di.setdefault(a, []).append(b)
    return di
Convert(words_freq, Prob_dist_dict)

#print(Prob_dist_dict)

robs = list(Prob_dist_dict.values())
```

```
flatProbs = [ item for elem in Probs for item in elem]
#print(Probs)
sum_prob = sum(flatProbs)
for x in range(len(Probs)):
    flatProbs[x] = flatProbs[x]/sum_prob

#print(flatProbs)

# Entropy calculation below
entropy = 0
for i in range(len(flatProbs)):
    entropy = entropy + (flatProbs[i]* math.log2(flatProbs[i]))
entropy = entropy*(-1)
print("The Shannon entropy is calculated to be: ",entropy)

# The block below returns n words chosen according to their
probability of appearance
# This is a "paragraph" of 100 words.
n=100
keys = np.array(list(Prob_dist_dict.keys()))
Probs = list(Prob_dist_dict.values())
flatProbs = [ item for elem in Probs for item in elem]
#print(Probs)
sum_prob = sum(flatProbs)
for x in range(len(Probs)):
    flatProbs[x] = flatProbs[x]/sum_prob
np.random.seed(2)
choice_list = np.random.choice(keys, n, replace=True, p=flatProbs)
print(choice_list)

# This block below tokenizes the word corpus
try: # Use the default NLTK tokenizer.
    from nltk import word_tokenize, sent_tokenize
    word_tokenize(sent_tokenize("This is a foobar sentence.
    Yes it is.")[0])
except: # Use a naive sentence tokenizer and toktok.
    import re
    from nltk.tokenize import ToktokTokenizer
    sent_tokenize = lambda x: re.split(r'(?<=[^A-Z].[.?])
    +(?=[A-Z])', x)
    toktok = ToktokTokenizer()
    word_tokenize = word_tokenize = toktok.tokenize


#This tokenizes our text saved in variable text
tokenized_text = [list(map(str.lower, word_tokenize(sent)))
                  for sent in sent_tokenize(str(text))]
```

```
# Preprocess the tokenized text for n-grams language modelling
n = 5
train_data, padded_sents = padded_everygram_pipeline(n,
tokenized_text)


model = MLE(n)
print("The n-gram model is training now...")
model.fit(train_data, padded_sents)
print("The model has been trained successfully. The details
are as follows:")
print(model.counts)

detokenize = TreebankWordDetokenizer().detokenize

def generate_sentence(model, num_words, random_seed=42):
    """
    :param model: An ngram language model from `nltk.lm.model`.
    :param num_words: Max no. of words to generate.
    :param random_seed: Seed value for random.
    """
    content = []
    for token in model.generate(num_words, random_seed=random_seed):
        if token == '<s>':
            continue
        if token == '</s>':
            break
        content.append(token)
    return detokenize(content)


for i in range(20):
    print("The random sentence number ",i," is: ")
    print(generate_sentence(model, 200, random_seed=i))
```

Code for problem 3

```
#This block reads the necessary input files
def load_data(data_path='data/'):
    train = os.path.join(data_path, "train.csv")
    test = os.path.join(data_path, "test.csv")
    return pd.read_csv(train), pd.read_csv(test)

train_df, test_df = load_data()

prices = pd.DataFrame({"price":train_df["SalePrice"], "log(price + 1)"
:np.log1p(train_df["SalePrice"])})
train_df["SalePrice"] = np.log1p(train_df["SalePrice"])
sns.distplot(train_df['SalePrice']);
```

```
# Drop Id column, and the target variable

train = train_df.drop('Id', axis=1)

test_ids = test_df['Id'].copy()
test = test_df.drop('Id', axis=1)

train.loc[:,'Train'] = 1
test.loc[:,'Train'] = 0

housing_df = pd.concat([train,test], ignore_index=True)

train_labels = train["SalePrice"].copy()
train = train.drop("SalePrice", axis=1) # drop labels for training set

#### Type of features

train['MSSubClass'] = train['MSSubClass'].astype(str)
test['MSSubClass'] = test['MSSubClass'].astype(str)

num_attribs = train.select_dtypes([np.number]).columns

cat_attribs = train.select_dtypes(include=[np.object]).columns

print('numerical:{} \n\n categorical:{}'.format(num_attribs, cat_attribs))

# Log transform the columns with high skew

skewed_cols = num_attribs[train[num_attribs].skew() > 0.75]
train[skewed_cols] = np.log1p(train[skewed_cols])
test[skewed_cols] = np.log1p(test[skewed_cols])

train.head(5)

train.isnull().sum()

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="mean")),
        ('std_scaler', StandardScaler()),
    ])

cat_pipeline = Pipeline([
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(handle_unknown='ignore',sparse=False)),
    ])
```

```
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", cat_pipeline, cat_attribs),
    ])

train_prepared = full_pipeline.fit_transform(train)


ridge_reg = Ridge(alpha=0.1)
ridge_reg.fit(train_prepared, train_labels)

train_predictions = ridge_reg.predict(train_prepared)
ridge_mse = mean_squared_error(train_labels, train_predictions)
ridge_rmse = np.sqrt(ridge_mse)
print('rmse on training:', ridge_rmse)

test_prepared = full_pipeline.transform(test)
test_predictions = ridge_reg.predict(test_prepared)
test_predictions = np.expm1(test_predictions)

def prep_to_submit(ids, preds, fname='submission.csv'):
    preds = pd.DataFrame({'Id': ids, 'SalePrice': preds})
    preds.to_csv(fname, index=False)

prep_to_submit(test_ids, test_predictions, fname='submission_ridge.csv')
print('this submission scored: ', 0.13636)

lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(train_prepared, train_labels)

train_predictions = lasso_reg.predict(train_prepared)
lasso_mse = mean_squared_error(train_labels, train_predictions)
lasso_rmse = np.sqrt(lasso_mse)
print('At alpha=0.1: ridge_rmse: {}, lasso_rmse: {}'
.format(ridge_rmse, lasso_rmse))

param_grid = [{'alpha': [0.0001, 0.0005, 0.001, 0.005, 0.01,
0.05, 0.1, 0.5, 1, 5, 10, 50, 100]}]

ridge_reg = Ridge()
lasso_reg = Lasso()

ridge_grid = GridSearchCV(ridge_reg, param_grid, cv=10,
scoring='neg_mean_squared_error', return_train_score=True)
lasso_grid = GridSearchCV(lasso_reg, param_grid, cv=10,
scoring='neg_mean_squared_error', return_train_score=True)

ridge_grid.fit(train_prepared, train_labels)
```

```
lasso_grid.fit(train_prepared, train_labels)

for mean_score, params in zip(ridge_grid.cv_results_
["mean_test_score"], ridge_grid.cv_results_["params"]):
    print(np.sqrt(-mean_score), params)

for mean_score, params in zip(lasso_grid.cv_results_
["mean_test_score"], lasso_grid.cv_results_["params"]):
    print(np.sqrt(-mean_score), params)

print('At tuned alphas - best scores: ridge_rmse: {},
lasso_rmse: {}'.format(min(np.sqrt(-ridge_grid.cv_results_
['mean_test_score'])), min(np.sqrt(-lasso_grid.cv_results_
['mean_test_score']))))

def display_scores(model, X, y, cv=10):
    scores = cross_val_score(model, X, y, n_jobs=-1,
    scoring='neg_mean_squared_error', cv=cv)
    print(str(model.__class__.__name__) + '; mean_rmse: {}'.
    format((np.sqrt(-scores)).mean()) + ', std_rmse: {}'.
    format((np.sqrt(-scores)).std()))

alphas = [1e-4, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1]
num_nonzeros = []
for alpha in alphas:
    lasso_reg = Lasso(alpha=alpha)
    lasso_reg.fit(train_prepared, train_labels)
    num_nonzeros.append(sum(lasso_reg.coef_>1e-3))

plt.plot(alphas, num_nonzeros)
plt.xlabel('alpha')
plt.ylabel('nonzero coefficient count')

estimators = [('r', ridge_grid),
              ('l', lasso_grid)
             ]

stacking_model = StackingRegressor(estimators=estimators,
passthrough=True)
#passthrough param if set true trains the final estimator
both on training data and prev predictors predictions
stacking_model.fit(train_prepared, train_labels);
display_scores(stacking_model, train_prepared, train_labels)

test_prepared = full_pipeline.transform(test)
test_predictions = stacking_model.predict(test_prepared)
test_predictions = np.expm1(test_predictions)
```

```
prep_to_submit(test_ids, test_predictions,
fname='submission_stacking_model.csv')
print('Score from the described stacking method in the problem:
', 0.13344)

param_distribs = {
        'max_depth': randint(low=1, high=20),
        'eta': [0.01, 0.05, 0.1, 0.5, 1],
        'subsample' : [0.1, 0.25, 0.5, 0.75, 1],
        'n_estimators' : randint(low=150, high=500),
    }

xgb_reg = xgboost.XGBRegressor(silent=True,
early_stopping_rounds=5)

rnd_search = RandomizedSearchCV(xgb_reg,
param_distributions=param_distribs,
                n_iter=20, cv=5, n_jobs=-1,
                scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(train_prepared, train_labels)


display_scores(rnd_search, train_prepared, train_labels)

#from part 1

def load_data(data_path='data/'):
    train = os.path.join(data_path, "train.csv")
    test = os.path.join(data_path, "test.csv")
    return pd.read_csv(train), pd.read_csv(test)

import joblib
def save_model(model, fname="model.pkl"):
    joblib.dump(model, fname)
def load_model(fname):
    return joblib.load(fname)

def display_scores(model, X, y, cv=10):
    scores = cross_val_score(model, X, y, n_jobs=-1,
    scoring='neg_mean_squared_error', cv=cv)
    print(str(model.__class__.__name__) + '; mean_rmse: {:.4f}
    w std ({:.4f})'.format((np.sqrt(-scores)).mean(),
    (np.sqrt(-scores)).std()))
    return scores

def prep_to_submit(ids, test, model):
    prepared = full_pipeline.transform(test)
    preds = model.predict(prepared)
```

```
    preds = np.expm1(preds)
    preds_df = pd.DataFrame({'Id': ids, 'SalePrice': preds})
    preds_df.to_csv(model.__class__.__name__ + '_preds.csv',
    index=False)


train_df, test_df = load_data()

test_ids = test_df['Id'].copy()

train_df.drop('Id', axis=1, inplace=True)
test_df.drop('Id', axis=1, inplace=True)



sns.distplot(train_df['SalePrice']);

prices = pd.DataFrame({"price":train_df["SalePrice"],
"log(price + 1)":np.log1p(train_df["SalePrice"])})
train_df["SalePrice"] = np.log1p(train_df["SalePrice"])
sns.distplot(train_df['SalePrice']);

corrmat = train_df.corr()
plt.subplots(figsize=(15,12));
sns.heatmap(corrmat, vmax=0.9, cmap="Blues", square=True);

#looks GarageCars and Garage Area are highly correlated
with each other, which is expected.
#looks like TotRmsAbvGrd and GrLivArea are highly correlated



corrmat['SalePrice'].sort_values(ascending=False)[:10]


#Let's investigate highly correlated features


cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars',
'TotalBsmtSF', '1stFlrSF', 'FullBath', 'YearBuilt']
sns.pairplot(train_df[cols], size = 2.5)
plt.show();

#here are 2 data points in GrLiveArea vs Sale Price that looks
like an outlier
#There is also another data point in TotalBsmtSF vs Sale price
that looks like an outlier
#Lets deal with these
```

```
plt.scatter(x = train_df['GrLivArea'], y = train_df['SalePrice'])
plt.ylabel('SalePrice')
plt.xlabel('GrLivArea')
plt.show()

#lets drop those 2 examples that looks a lot like outliers



train_df = train_df.drop(train_df[(train_df['GrLivArea']>4000)
& (train_df['SalePrice']<12.5)].index)



plt.scatter(x = train_df['TotalBsmtSF'], y = train_df['SalePrice'])
plt.ylabel('SalePrice')
plt.xlabel('TotalBsmtSF')
plt.show()

#That data point looked like an outlier to TotalBsmtSF was the
same point we just dropped - so we are good

# feature OverallQual
plt.subplots(figsize=(8, 6))
sns.boxplot(x=train_df['OverallQual'], y=train_df["SalePrice"])

#Lets treat this as a category rather than a numerical value

train_df['OverallCond'] = train_df['OverallCond'].astype(str)
test_df['OverallCond'] = test_df['OverallCond'].astype(str)

plt.scatter(x = train_df['TotalBsmtSF'], y = train_df['SalePrice'])
plt.ylabel('SalePrice')
plt.xlabel('TotalBsmtSF')
plt.show()

train_labels = train_df['SalePrice'].copy()

train = train_df.drop('SalePrice', axis=1)
test = test_df.copy()



cat_attribs = train.select_dtypes(include=[np.object]).columns
train[cat_attribs].describe(include='all')

train['Street'].value_counts()
```

```
train['Utilities'].value_counts()

#lets drop the above two categorical features, there is no
variation no information here to learn from



train.drop(columns=['Utilities', 'Street'], inplace=True)
test.drop(columns=['Utilities', 'Street'], inplace=True)



total = train_df.isnull().sum().sort_values(ascending=False)
percent = (train_df.isnull().sum()/train_df.isnull().count())
.sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1,
keys=['Total', 'Percent'])
missing_data.head(20)

# From the description of data MSSubClass looks like a categorical value
train[['MSSubClass', 'YrSold', 'MoSold']] =
train[['MSSubClass', 'YrSold', 'MoSold']].astype(str)
test[['MSSubClass', 'YrSold', 'MoSold']] =
test[['MSSubClass', 'YrSold', 'MoSold']].astype(str)


#From: got ideas from https://www.kaggle.com/niteshx2/
top-50-beginners-stacking-lgb-xgb

def fix_nas(df):
    #Fill na for these column with standard equipment
    for those - intuition from data description.txt
    df['Functional'] = df['Functional'].fillna('Typ')
    df['Electrical'] = df['Electrical'].fillna('SBrkr')
    df['KitchenQual'] = df['KitchenQual'].fillna('TA')
    df['SaleType'] = df['SaleType'].fillna('Other')
    df['Exterior1st'] = df['Exterior1st'].fillna('Other')
    df['Exterior2nd'] = df['Exterior2nd'].fillna('Other')

    #None for not exists - intuition from data description.txt
    df['PoolQC'] = df['PoolQC'].fillna('None')

    #These two are probably very related
    df['MSZoning'] = df.groupby('MSSubClass')['MSZoning']
    .transform(lambda x: x.fillna(x.mode()[0]))

    #Na means No Garage
    for col in ['GarageYrBlt', 'GarageArea', 'GarageCars']:
```

```
        df[col] = df[col].fillna(0)

    #Na means No Garage
    for col in ['GarageType', 'GarageFinish', 'GarageQual'
    , 'GarageCond']:
        df[col] = df[col].fillna('None')

    #Na means there's no basement
    for col in ['BsmtQual', 'BsmtCond', 'BsmtExposure',
    'BsmtFinType1', 'BsmtFinType2']:
        df[col] = df[col].fillna('None')

    #Na means no basement, so the measurement is 0
    for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
    'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath'):
        df[col] = df[col].fillna(0)

    for col in train.select_dtypes(include=[np.object])
    .columns:
        df[col] = df[col].fillna('None')

    for col in train.select_dtypes(include=[np.number])
    .columns:
        df[col] = df[col].fillna(0)

    return df

train_df = fix_nas(train_df)
test_df = fix_nas(test_df)

num_attribs = train.select_dtypes([np.number]).columns

skewed_cols = num_attribs[train[num_attribs].skew() > 0.7]
train[skewed_cols] = np.log1p(train[skewed_cols])
test[skewed_cols] = np.log1p(test[skewed_cols])

def combine_features(df):

    # Exists or not
    df['HasPool'] = df['PoolArea'].apply(lambda x: 1
    if x > 0 else 0)
    df['Has2ndfloor'] = df['2ndFlrSF'].apply(lambda x: 1
    if x > 0 else 0)
    df['HasGarage'] = df['GarageArea'].apply(lambda x: 1
    if x > 0 else 0)
    df['HasBsmt'] = df['TotalBsmtSF'].apply(lambda x: 1
    if x > 0 else 0)
    df['HasFireplace'] = df['Fireplaces'].apply(lambda x:
```

```
    1 if x > 0 else 0)

    #Unfinished or not
    df['BsmtFinType1_Unf'] = 1*(df['BsmtFinType1'] ==
    'Unf')

    df['OldHouse'] = df['YearBuilt'].apply(lambda x: 1
    if x <1990 else 0)

    #Some aggregated features
    df['TotalSF']=df['TotalBsmtSF'] + df['1stFlrSF'] +
    df['2ndFlrSF']
    df['TotalBathrooms'] = (df['FullBath'] +
    (0.5 * df['HalfBath']) + df['BsmtFullBath'] +
    (0.5 * df['BsmtHalfBath']))
    df['TotalPorchSF'] = (df['OpenPorchSF']
    + df['3SsnPorch'] + df['EnclosedPorch']
    + df['ScreenPorch'] +  df['WoodDeckSF'])



    df['Age_YrBuilt'] = df['YrSold'] - df['YearBuilt']
    df['Age_YrRemod'] = df['YrSold'] - df['YearRemodAdd']
    df['Age_Garage'] = df['YrSold'] - df['GarageYrBlt']
    df['Remodeled'] = df['YearBuilt']!=df['YearRemodAdd']

    #to fix if the garageyrbuilt is 0 for example
    - its 0 if garage was never built
    df['Age_YrBuilt'] = df['Age_YrBuilt'].apply(lambda x: 0
    if x <0 else x)
    df['Age_YrRemod'] = df['Age_YrRemod'].apply(lambda x: 0
    if x <0 else x)
    df['Age_Garage'] = df['Age_Garage'].apply(lambda x: 0
    if x <0 else x)


    return df

train_df = combine_features(train_df)
test_df = combine_features(test_df)

corrmat = train_df.corr()
corrmat['SalePrice'].sort_values(ascending=False)[:25]

corrmat['SalePrice'].sort_values(ascending=True)[:25]

#Lets drop the two least correlated features
train.drop(columns=['BsmtFinSF2', 'BsmtHalfBath'],
```

```
inplace=True)
test.drop(columns=['BsmtFinSF2', 'BsmtHalfBath'],
inplace=True)

#### Type of features
num_attribs = train.select_dtypes([np.number]).columns

ord_attribs = list(['FireplaceQu', 'BsmtQual', 'BsmtCond',
'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC',
        'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'GarageFinish',
        'LandSlope', 'YrSold', 'MoSold'])

cat_attribs = train.select_dtypes(include=[np.object]).columns

print('numerical:{} \n\n categorical:{}'
.format(num_attribs, cat_attribs))

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="mean")),
        ('std_scaler', RobustScaler()),
    ])

# ord_pipeline = Pipeline([
#         ("imputer", SimpleImputer(strategy="most_frequent")),
#         ("encoder", OrdinalEncoder()),
#     ])

cat_pipeline = Pipeline([
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(handle_unknown='ignore',sparse=False)),
    ])

full_pipeline = ColumnTransformer([
#         ("ord", ord_pipeline, ord_attribs),
        ("num", num_pipeline, num_attribs),
        ("cat", cat_pipeline, cat_attribs),
    ])

train_prepared = full_pipeline.fit_transform(train)

train_prepared.shape

ridge_reg = Ridge(max_iter=3000)
lasso_reg = Lasso(max_iter=3000)

param_grid = [{'alpha': [0.0001, 0.0005, 0.001, 0.005, 0.01,
```

```
0.05, 0.1, 0.5, 1, 5, 10, 50]}]

ridge_grid = GridSearchCV(ridge_reg, param_grid, cv=10,
scoring='neg_mean_squared_error', return_train_score=True)
lasso_grid = GridSearchCV(lasso_reg, param_grid, cv=10,
scoring='neg_mean_squared_error', return_train_score=True)

ridge_grid.fit(train_prepared, train_labels)
lasso_grid.fit(train_prepared, train_labels)

lasso_scores = display_scores(lasso_grid.best_estimator_,
train_prepared, train_labels)
ridge_scores = display_scores(ridge_grid.best_estimator_,
train_prepared, train_labels)

gbrt = GradientBoostingRegressor(min_samples_leaf=6,
max_depth=4, max_features='sqrt', subsample=0.8, warm_start=True)
X_train, X_val, y_train, y_val = train_test_split(train_prepared,
train_labels)

min_val_error = np.inf
err_up = 0
for n_estimators in range(1,1000):
    gbrt.n_estimators = n_estimators
    gbrt.fit(X_train, y_train)
    y_pred = gbrt.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
        err_up = 0
    else:
        err_up += 1
        if err_up == 5:
            break

gbrt_scores = display_scores(gbrt, train_prepared, train_labels)

param_grid = [
#         {'kernel': ['linear'], 'C': [0.1, 1, 10, 20]},
        {'kernel': ['rbf'], 'C': [0.01, 1.0, 3.0, 10.0,
        30.0, 100.0, 300.0, 1000.0, 3000.0],
        'gamma': [1e-6, 0.00001, 0.0003, 0.0001, 0.003,
        0.01, 0.03, 0.1]},
    ]

svm_reg = SVR()
svm_grid = GridSearchCV(svm_reg, param_grid, cv=10,
scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)
svm_grid.fit(train_prepared, train_labels)
```

```
svm_scores = display_scores(svm_grid.best_estimator_,
train_prepared, train_labels, cv=10)
svm_grid.best_params_

param_grid = {
        'n_estimators': [50, 100, 250, 500, 1000],
        'max_features': [32, 64, 128],
        'max_depth': [2, 4, 8, 16, 32],
        'bootstrap': [False, True],
#         'max_samples': [0.25, 0.5, 0.75],
        'min_samples_leaf': [2, 3, 5, 8],
        'min_samples_split' : [2, 4, 8]
    }



rf_reg = RandomForestRegressor(random_state=42)
rf_grid = GridSearchCV(rf_reg, param_grid=param_grid, cv=4,
scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)


rf_grid.fit(train_prepared, train_labels)

rf_grid.best_params_

rf_scores = display_scores(rf_grid.best_estimator_,
train_prepared, train_labels, cv=10)

#https://towardsdatascience.com/fine-tuning-xgboost-in
-python-like-a-boss-b4543ed8b1e
param_distribs = {
        'max_depth': randint(low=3, high=20),
        'eta': [0.01, 0.05, 0.1],
        'subsample' : [0.8, 1],
        'colsample_bytree' : [0.3, 0.5, 0.8],
        'n_estimators' : randint(low=400, high=1000),
        'min_child_weight' : np.arange(1,6,2)
    }

xgb_reg = xgboost.XGBRegressor(silent=True,
early_stopping_rounds=5)

xgb_grid = RandomizedSearchCV(xgb_reg,
param_distributions=param_distribs,  n_iter=50, cv=5,
n_jobs=-1, scoring='neg_mean_squared_error', random_state=42)
xgb_grid.fit(train_prepared, train_labels);
```

```
xgb_scores = display_scores(xgb_grid.best_estimator_,
train_prepared, train_labels, cv=5)



xgb_grid.best_params_

xgb_final = xgboost.XGBRegressor(early_stopping=7)

estimators = [('r', ridge_grid.best_estimator_),
              ('l', lasso_grid.best_estimator_),
              ('boost', gbrt),
              ('svm', svm_grid.best_estimator_),
              ('rf', rf_grid.best_estimator_),
              ('xgb', xgb_grid.best_estimator_)
             ]

stack = StackingRegressor(estimators=estimators,
n_jobs=-1, passthrough=True)
stack_2 = StackingRegressor(estimators=estimators,
n_jobs=-1, passthrough=False)

display_scores(stack, train_prepared, train_labels)
stack_scores = display_scores(stack_2,
train_prepared, train_labels)
stack_2.fit(train_prepared, train_labels);

def blended_predictions(X):
    return ((0.1 * ridge_grid.best_estimator_.predict(X)) + \
            (0.2 * lasso_grid.best_estimator_.predict(X)) + \
            (0.1 * gbrt.predict(X)) + \
            (0.1 * xgb_grid.best_estimator_.predict(X)) + \
            (0.1 * svm_grid.best_estimator_.predict(X)) + \
            (0.05 * rf_grid.best_estimator_.predict(X)) + \
            (0.35 * stack_2.predict(X)))

blended_mse = mean_squared_error(train_labels,
blended_predictions(train_prepared))
blended_score = np.sqrt(blended_mse)
print('RMSLE score on train data:')
print(blended_score)



model = stack_2

model.fit(train_prepared, train_labels)
prepared = full_pipeline.transform(test)
```

```
# preds = model.predict(prepared)
preds = blended_predictions(prepared)
preds_transformed = np.expm1(preds)
preds_df = pd.DataFrame({'Id': test_ids,
'SalePrice': preds_transformed})
preds_df.to_csv('submission_' + model.__class__.__name__ +
'.csv', index=False)

model.fit(train_prepared, train_labels)
# train_preds = model.predict(train_prepared)
train_preds = blended_predictions(train_prepared)
mse = mean_squared_error(train_labels, train_preds)
rmse = np.sqrt(mse)
rmse

models = [ridge_grid, lasso_grid, xgb_grid,
svm_grid, gbrt, rf_grid, stack]
for model in models:
    save_model(model, fname=model.__class__.__name__ + '.pkl')
```