Lab 3

Abhishek Sridhar
Connor Smith
Tom Li

Q1: Summary

Information can be compressed - represented with less than the original amount of bits - without loss of information. There is a mathematical limit on the amount of data compression that can be done without losing information. It is H(x) = - Sum(p(x) * log(p(x))) where p(x) is the probability of the random variable. The author mentions several approximation methods used in sentence information compression. The N-gram method is called. Words are grouped into N word groups and for each group a probability is produced to estimate the probability of the next word being a specific word given the previous N-1 words. For example, P(w1, w2, …, wn-1, wn) will give the probability that the next word is wn given the previous n-1 words are w1, w2, …, wn-1. This has a connection with a stochastic process where previous states will influence subsequent states. The stochastic process can be represented in a graphical format where each state is a dot and there are arrows connecting the states with probabilities of the next states. There are at most n^2 in a trigram Markov stochastic diagram. Besides stochastic processes, there is also ergodic process. Ergodic process produces sequences that have the same statistical properties: letter frequencies, diagram frequencies. Entropy is given by the formula above and there are some interesting observations from that formula. When p=0 or 1, H=0. Meaning when the value is completely certain the entropy of the system is 0. Conversely, entropy is maximized to 1 when p=½ where there is no predictive advantage using probability. Relative entropy is the maximum value of a source while still staying with the same alphabet code. Relative entropy has one drawback - it has redundancy when compared to normal entropy because of conforming to the existing alphabet code which might not be optimal.

```python
import math
import os
from collections import Counter
from random import choices
from urllib.parse import urljoin

import bs4
import requests
import textract

# part 1
url = "http://proceedings.mlr.press/v70/"

folder_location = r'p2files'
if not os.path.exists(folder_location): os.mkdir(folder_location)

response = requests.get(url)
soup = bs4.BeautifulSoup(response.text, "html.parser")

for link in soup.select("a[href$='.pdf']"):
    #Name the pdf files using the last portion of each link which are unique in this case
    filename = os.path.join(folder_location,link['href'].split('/')[-1])
    with open(filename, 'wb') as f:
        print(f)
        f.write(requests.get(urljoin(url,link['href'])).content)


text = ''
for filename in os.listdir('p2files'):
    size = os.path.getsize('p2files/' + filename)
    if size > 0:
        text += textract.process('p2files/' + filename).decode('utf-8')

text = text.replace('\x00', '')
text = text.split()

Counters_found = Counter(text)
most_occur = Counters_found.most_common(10)
```

```python
print(most_occur)

# part 2
all_words = list(Counter(text).items())

total_word_count = sum(Counters_found.values())
entropy_sum = 0

# entropy
for i in all_words:
    entropy_sum -= (i[1] / total_word_count) * math.log((i[1] / total_word_count), 2)

print(entropy_sum)

# part 3
words = []
weights = []

for i in Counters_found.keys():
    words.append(i.replace('.', ''))

for i in Counters_found.values():
    weights.append(i / total_word_count)

random_paragraph = ''
for i in range(100):
    if i % 10 == 0 and (i > 0):
        random_paragraph += '. '
        random_paragraph += choices(words, weights)[0].capitalize()

    elif i % 10 != 0:
        random_paragraph += ' '
        random_paragraph += choices(words, weights)[0]

    elif (i % 10) == 0 and i == 0:
        random_paragraph += choices(words, weights)[0].capitalize()

print(random_paragraph)

```

```python
import matplotlib
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import xgboost as xgb

from scipy.stats import skew
from collections import OrderedDict
# The error metric: RMSE on the log of the sale prices.
from sklearn.metrics import mean_squared_error

def rmse(y_true, y_pred):
        return np.sqrt(mean_squared_error(y_true, y_pred))

def Q3_P1():
    # ===========================================================================
    # read in the data
    # ===========================================================================
    train_data = pd.read_csv('../input/train.csv', index_col=0)
    test_data = pd.read_csv('../input/test.csv', index_col=0)


    # ===========================================================================
    # here, for this simple demonstration we shall only use the numerical columns
    # and ingnore the categorical features
    # ===========================================================================
    X_train = train_data.select_dtypes(include=['number']).copy()
    X_train = X_train.drop(['SalePrice'], axis=1)
    y_train = train_data["SalePrice"]
    X_test = test_data.select_dtypes(include=['number']).copy()
```

```

    regressor = xgb.XGBRegressor()


# ===========================================================================
    # exhaustively search for the optimal hyperparameters
# ===========================================================================
    from sklearn.model_selection import GridSearchCV
    # set up our search grid
    param_grid = {"max_depth": [3, 4],
                "n_estimators": [ 600, 700],
                "learning_rate": [0.015, 0.020, 0.025]}


    # try out every combination of the above values
    search = GridSearchCV(regressor, param_grid, cv=5).fit(X_train, y_train)

    print("The best hyperparameters are ", search.best_params_)

    regressor = xgb.XGBRegressor(learning_rate=search.best_params_["learning_rate"],
                                 n_estimators=search.best_params_["n_estimators"],
                                max_depth=search.best_params_["max_depth"])


    regressor.fit(X_train, y_train)


# ===========================================================================
    # use the model to predict the prices for the test data
# ===========================================================================
```

```
==========================
    predictions = regressor.predict(X_test)
    # read in the ground truth file
    solution = pd.read_csv('../input/solution.csv')
    y_true = solution["SalePrice"]

    from sklearn.metrics import mean_squared_log_error
    RMSLE = np.sqrt(mean_squared_log_error(y_true, predictions))
    print("The score of forum post is %.5f" % RMSLE )
# ===========================================================================
    # write out CSV submission file
# ===========================================================================
    output = pd.DataFrame({"Id": test_data.index, "SalePrice": predictions})
    output.to_csv('forum_submission.csv', index=False)

    """What I Learned
    I learned about using CV to optimize parameters. More specifically I learned about the GridSearch CV
    which can optimize hyperparameters for any model. Additionally, I gained a greater understand of XGBoost and how to use it"""


def Q3_P2():
    # PREPROCESSING

    train = pd.read_csv("../input/train.csv")
    test = pd.read_csv("../input/test.csv")
    train.head()
    all_data = pd.concat((train.loc[:, 'MSSubClass':'SaleCondition'],
                          test.loc[:, 'MSSubClass':'SaleCondition']))
    matplotlib.rcParams['figure.figsize'] = (12.0,
```

```python
6.0)
    prices = pd.DataFrame({"price": train["SalePrice"], "log(price + 1)": np.log1p(train["SalePrice"])})
    # prices.hist()

    # log transform the target:
    train["SalePrice"] = np.log1p(train["SalePrice"])

    # log transform skewed numeric features:
    numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

    skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))  # compute skewness
    skewed_feats = skewed_feats[skewed_feats > 0.75]
    skewed_feats = skewed_feats.index

    all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

    all_data = pd.get_dummies(all_data)

    # filling NA's with the mean of the column:
    all_data = all_data.fillna(all_data.mean())

    # creating matrices for sklearn:
    X_train = all_data[:train.shape[0]]
    X_test = all_data[train.shape[0]:]
    y = train.SalePrice

    regressor = xgb.XGBRegressor()


    # ===========================================================================
    # exhaustively search for the optimal hyperparameters
    # ===========================================================================
    from sklearn.model_selection import
```

```python
GridSearchCV
    # set up our search grid
    param_grid = {"max_depth": [3, 4],
                  "n_estimators": [700, 1000, 5000],
                  "learning_rate": [0.01, 0.04, .05]}

    # try out every combination of the above values
    #search = GridSearchCV(regressor, param_grid, cv=5).fit(X_train, y)

    #print("The best hyperparameters are ", search.best_params_)

    #regressor = xgb.XGBRegressor(learning_rate=search.best_params_["learning_rate"],
    #                             n_estimators=search.best_params_["n_estimators"],
    #                             max_depth=search.best_params_["max_depth"])
    regressor = xgb.XGBRegressor(learning_rate=.01,
                                 n_estimators=5000,
                                 max_depth=4)

    regressor.fit(X_train, y)


# ===========================================================================
    # use the model to predict the prices for the test data
# ===========================================================================
    predictions = regressor.predict(X_test)
    # read in the ground truth file
    solution = pd.read_csv('../input/solution.csv')
    y_true = solution["SalePrice"]

```

```python
    from sklearn.metrics import mean_squared_log_error
    RMSLE = np.sqrt(mean_squared_log_error(y_true, np.expm1(predictions)))
    print("The score of improved forum post is %.5f" % RMSLE)
# ===========================================================================
    # write out CSV submission file
# ============================================================================
    output = pd.DataFrame({"Id": test.Id, "SalePrice": np.expm1(predictions)})
    output.to_csv('improved_forum_submission.csv', index=False)
    """Our Approach:
    Our approach was to improve the test score was to add data preprocessing steps, and to tune the hyperparameters.
    We tried many different value combinations when using GridSearchCV to find the best hyperparameters.
    For preprocessing, we used a log transform on the numerical data while also replacing NA values with the mean."""



def Q3_P3():
    # PREPROCESSING

    train = pd.read_csv("../input/train.csv")
    test = pd.read_csv("../input/test.csv")
    train.head()
    all_data = pd.concat((train.loc[:, 'MSSubClass':'SaleCondition'],
                          test.loc[:, 'MSSubClass':'SaleCondition']))
    matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
    prices = pd.DataFrame({"price": train["SalePrice"], "log(price + 1)": np.log1p(train["
```

```python
SalePrice"])})
    # prices.hist()

    # log transform the target:
    train["SalePrice"] = np.log1p(train["SalePrice"])

    # log transform skewed numeric features:
    numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

    skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))  # compute skewness
    skewed_feats = skewed_feats[skewed_feats > 0.75]
    skewed_feats = skewed_feats.index

    all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

    all_data = pd.get_dummies(all_data)

    # filling NA's with the mean of the column:
    all_data = all_data.fillna(all_data.mean())

    # creating matrices for sklearn:
    X_train = all_data[:train.shape[0]]
    X_test = all_data[train.shape[0]:]
    y = train.SalePrice

    # Models
    from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, Lasso, LassoLarsCV
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import mean_squared_error

    #UNDERFITTING MODEL
    model_lasso = Lasso(1000)
    model_lasso.fit(X_train, y)
    l1_pred = model_lasso.predict(X_test)
    l1_pred_train = model_lasso.predict(X_train)
    rms = np.sqrt(mean_squared_error(y, l1_pred_train))
```

```python
    print("Lasso Underfit Training Score:", rms)
    solution = pd.DataFrame({"id": test.Id, "SalePrice": np.expm1(l1_pred)})
    solution.to_csv("Lasso_Regression_Underfit.csv", index=False)

    #OVERFITTING MODEL

    dtrain = xgb.DMatrix(X_train, label=y)
    dtest = xgb.DMatrix(X_test)

    params = {"max_depth": 100, "eta": 0.1}
    model_xgb = xgb.XGBRegressor(n_estimators=360, max_depth=100, learning_rate=0.1)  # the params were tuned using xgb.cv
    model_xgb.fit(X_train, y)
    xgb_preds = np.expm1(model_xgb.predict(X_test))
    xgb_preds_train = model_xgb.predict(X_train)

    solution = pd.DataFrame({"id": test.Id, "SalePrice": xgb_preds})
    solution.to_csv("XGB_Regression.csv", index=False)

    print("XGBoost Overfit Training Score:", rmse(y, xgb_preds_train))





# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    Q3_P1()
    Q3_P2()
    Q3_P3()
```