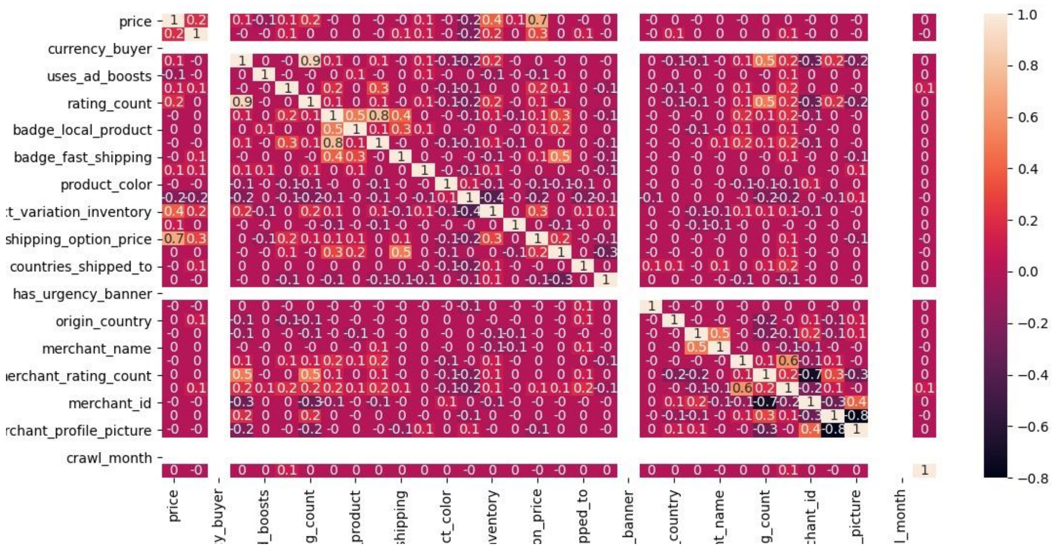
Jacqueline Heaton 16jh12 - A0

Data Exploration



After imputation and encoding, the correlation map for all 34 variables can be seen, and the ones relevant to the rating manually picked out. Picking a threshold – for example, 0.1, for which the absolute correlation must equal or exceed allows for relevant variables to be identified. The variables with an absolute correlation >= 0.1 include price, badges_count, badge_product_quality, shipping_option_price, inventory_total, and merchant_rating. Some of these may only be correlated coincidently, and so we may wish to use our own judgement for including/excluding certain variables.

Problem Formulation

The goal is to predict the rating of a product based on other variables. As such, the input will be the variables for the product that we think contribute to the rating score, and the output will be the predicted rating. This will require preprocessing (imputation and encoding, scaling, normalizing, etc), extracting validation data, and then an AI model tweaked to suit the input data. Challenges include finding an appropriate means of preprocessing, and the correct parameters for the AI models.

Documentation:

*** Unless stated otherwise, the scores given are the f1 scores of the validation data**

Multi-Layer Perceptron

- Started with layers (10, 5), a = 1e-3, solver = lbfgs, k = 4 (for cross validation)

  - Tends to go from high 60s to mid 70s
- Changed solver = sgd – lbfgs was not particularly good, sgd is based on what we learned in class
  - No change
  - Added activation = logistic – trying different activation functions
    - High 60s to mid 70s
  - Changed activation = tanh
    - High 60s to low 70s
  - Got rid of activation (default=relu), changed layers to (50,10)
    - High 60s to low 70s
  - Layers (100,10)
    - High 60s to low 70s
- CHANGED K = 8
  - Solver = sgd, layers (100, 10)
    - Mid 60s – high 70s
- Increased the columns from which data is used – low numbers may be due to not enough data, and correlation may be relevant, even if small
  - K = 5, solver = sgd, layers = (120,20), learning rate = 'adaptive'
    - High 60s – mid 70s
  - Changed learning rate back to default, changed iter_no_change to 15 (from default of 10) and tol to 1e-5 (from default 1e-4)
    - Mid 60s – low 70s

Random Forest

- Started with 100 estimators, max_depth = 15, k = 4
  - Tends to go from low 70s to high 70s
- Reduced estimators to 10 – to see what min number of estimators is
  - Tends to go from high 60s to mid 70s
- Set estimators to 50 – 10 was too low
  - Tends to be in 70s range
  - Got rid of max_depth – allows to go until all leaves pure
    - High 60s to mid 70s
- Set estimators to 20, still no max_depth
  - High 60s to mid 70s
- Set estimators to 70
  - High 60s to mid 70s
  - Interesting, since did better with 50 estimators. Likely just chance
- Set estimators to 120
  - High 60s to mid 70s

- CHANGED K = 8
    - Estimators 120
        - High 60s to high 70s
- Increased the columns from which data is used – low numbers may be due to not enough data, and correlation may be relevant, even if small
    - Set everything to default
        - High 70s to low 80s when only encoding
        - High 60s to mid 70s when normalizing or scaling
- Checked f1_score of training data, all 99%
    - Highest seen yet

Support Vector Machine

- Started with kernel = poly, degree = 2
    - Tends to go from high 60s to mid 70s
- Changed to kernel = rbf – trying out different kernels to see which works best
    - No change
- Changed kernel = linear
    - No change
- Changed kernel = sigmoid
    - High 50s to mid 60s
- Changed kernel = poly, degree = 3
    - High 60s to mid 70s
    - Changed degree = 4 – trying to find where overfitting occurs
        - High 60s to mid 70s
    - Changed degree = 5
        - High 60s to mid 70s
    - Changed degree = 8
        - High 60s to mid 70s
- CHANGED K = 8
    - Degree = 8
        - Mid 60s to mid 70s
- Increased the columns from which data is used – low numbers may be due to not enough data, and correlation may be relevant, even if small
    - Poly, degree 3
        - Low 60s to mid 70s

Questions

Why Data Mining is a misnomer? What is another preferred name?

Data mining refers to the process of acquiring useful knowledge from massive amounts of data, whereas mining implies taking any and all data in its raw form, regardless of how useful it is. Another name is knowledge discovery or extraction.

What is the general knowledge discovery process? What is the difference between a data engineer and data scientist/AI engineer?

The general knowledge discovery process is the process by which data in analyzed and used. This requires acquiring the data, cleaning the data, identifying the task-relevant data, and then analyzing for any patterns and turning that into knowledge.

A data scientist follows the proper hierarchy of needs for collecting, cleaning, transforming, and using data for AI and deep learning. Data engineers are also involved in collecting data, but don't necessarily clean and transform it.

In data mining, what is the difference between prediction and categorization?

Prediction is for numerical data, where a number is predicted for an input – such as predicting the price of an item. Categorization is when the input is classified as belonging to a category, like cat or dog.

In a linear model, which regularization method encourages sparsity?

The Lasso regularization.

Why we need GD for optimization, rather than simply use linear algebra to solve a simple linear model?

Gradient descent is computationally easier/cheaper than using linear algebra in some cases, which becomes important when the amount of data and the complexity of the linear model increases. Gradient descent requires less memory too, whereas using linear algebra requires all the data to be held in memory, which may not be feasible.

In terms of bias and variance, defines what is overfitting and under fitting?

Overfitting means the model it too precisely matching the training data, but misses the overall pattern – in other words, it is biased. Underfitting means the model is not precise enough, and would result in high variance.

Why data science/machine learning is a bad idea in the context of information security?

Machine learning requires massive amounts of data, often personal data collected without the conscious consent of the people – recordings of the ads people clicked, videos they watched, places they go, etc. – and in the event of a data breach, this personal information could cause serious harm. Similarly, this may be the case even without a data leak, as the AI models may use any and all information they have access to, even if it isn't relevant to the classification/prediction. This may cause patterns to be unknowingly included in the decisions which may reveal information to an observant spectator, or someone who uses the model later on.

Code

```
# CISC 873
# Data Mining
# Assignment 0
# Jacqueline Heaton
# 20028278
# 16jh12

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import svm
from sklearn import preprocessing


def impute_encode(column):
    # replace missing vals with most common value - including strings
    mostFreq = column.mode()

    # for when all have the same number of appearances
    if len(mostFreq) > 1:
        mostFreq = 0 # should only be issue in ids, which isnt relevant anyway
    column = column.replace([np.nan], mostFreq)

    # convert to type category
    column = column.astype('category')
    # convert all into numbers, including strings
    column = column.cat.codes
    return column

# displays heat map with correlation values
# takes in dataframe, converts all string columns to numbers
def heatmap(trainData):
    # must impute and encode to include string data
    cols = [i for i in trainData.columns]
    for i in cols:
        trainData[i] = impute_encode(trainData[i])
    # get correlation matrix
    trainData.corr().to_csv("Correlation.csv",index=False)

    # according to heatmap, values w correlation > 0.2 = price, badges_count,
badge_product_quality, merchant_rating
    plt.figure()
    sns.heatmap(data=trainData.corr().round(1), annot=True)
    plt.show()


# selects columns of relevance (correlation >0.1)
# replaces NaN values with the mean
# removes outliers
def preprocess1(train, test, k):

    data =
train.drop(['currency_buyer','rating','has_urgency_banner','urgency_text','badge_loca
l_product',

'merchant_id','theme','crawl_month','id','merchant_profile_picture','merchant_rating_
count',

'merchant_has_profile_picture','merchant_name','merchant_title','countries_shipped_to
',

'product_variation_inventory','product_variation_size_id','tags'], axis=1)
    # select relevent columns
    # data = train.loc[:,['price',
'uses_ad_boosts','rating_count','countries_shipped_to','units_sold','badge_local_prod
uct',
    #          'has_urgency_banner','product_color',
'product_variation_inventory','merchant_info_subtitle','origin_country','retail_price
','badge_fast_shipping','badges_count',
    #          'badge_product_quality', 'shipping_option_price',
'merchant_rating','merchant_rating_count','inventory_total']]
    # data = train.loc[:,
    #         ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_rating', 'inventory_total']]
    # pull ids and ratings for saving to file
    ids = train.loc[:,'id']
    ans = train.loc[:,'rating']

    testData = test.drop(['currency_buyer', 'has_urgency_banner', 'urgency_text',
'badge_local_product',
                          'merchant_id', 'theme', 'crawl_month', 'id',
'merchant_profile_picture','merchant_rating_count',
                          'merchant_has_profile_picture', 'merchant_name',
'merchant_title', 'countries_shipped_to',

'product_variation_inventory','product_variation_size_id','tags'], axis=1)
    # testData = test.loc[:,['price',
'uses_ad_boosts','rating_count','countries_shipped_to','units_sold','badge_local_prod
uct',
    #          'has_urgency_banner', 'product_color',
'product_variation_inventory','merchant_info_subtitle','origin_country','retail_price
','badge_fast_shipping','badges_count',
    #          'badge_product_quality', 'shipping_option_price',
'merchant_rating','merchant_rating_count','inventory_total']]
    # testData = test.loc[:,
    #            ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_rating', 'inventory_total']]
    testIds = test.loc[:, 'id']

    # impute and encode
    cols = [i for i in data.columns]
    for i in cols:
        data[i] = impute_encode(data[i])

    cols = [i for i in testData.columns]
    for i in cols:
        testData[i] = impute_encode(testData[i])

    # does cross validation
    # returns trainData, trainAns, validationData, validationAns, validationIds
    trainData, trainAns, validData, validAns, validIds = getValidation(data, ans,
ids, k)

    return trainData, trainAns, validData, validAns, validIds, testData, testIds


# selects columns of relevance (correlation >0.1)
# normalizes data
def preprocess2(train, test, k):
    data = train.loc[:,
           ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price',
            'merchant_rating', 'inventory_total']]

    # data = train.loc[:,
    #        ['price',
'uses_ad_boosts','rating_count','countries_shipped_to','units_sold','badge_local_prod
uct',
    #
'has_urgency_banner','origin_country','retail_price','badge_fast_shipping','badges_co
unt',
    #         'badge_product_quality', 'shipping_option_price',
'merchant_rating','merchant_rating_count','inventory_total']]

    # pull ids and ratings for saving to file
    ids = train.loc[:, 'id']
    ans = train.loc[:, 'rating']

    testData = test.loc[:,
           ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price',
            'merchant_rating', 'inventory_total']]

    # testData = test.loc[:,['price',
'uses_ad_boosts','rating_count','countries_shipped_to','units_sold','badge_local_prod
uct',
    #
'has_urgency_banner','origin_country','retail_price','badge_fast_shipping','badges_co
unt',
    #         'badge_product_quality', 'shipping_option_price',
'merchant_rating','merchant_rating_count','inventory_total']]


    testIds = test.loc[:, 'id']

    # normalize data as preprocessing step
    trainData = pd.DataFrame(preprocessing.normalize(data))
    testData = pd.DataFrame(preprocessing.normalize(testData))

    # normalize train data before sending in for validation
    trainData, trainAns, validData, validAns, validIds = getValidation(trainData,
ans, ids, k)


    return trainData, trainAns, validData, validAns, validIds, testData, testIds

# selects columns of relevance (correlation >0.1)
# scales data
def preprocess3(train, test, k):
    data = train.loc[:,
           ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_rating',
            'inventory_total']]
    # data = train.loc[:,
    #        ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_info_subtitle',
    #         'merchant_rating', 'inventory_total']]

    ids = train.loc[:, 'id']
    ans = train.loc[:, 'rating']

    testData = test.loc[:,
               ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_rating',
                'inventory_total']]

    # testData = test.loc[:,
    #        ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_info_subtitle',
    #         'merchant_rating', 'inventory_total']]

    testIds = test.loc[:, 'id']

    # preprocess using scale
    data = pd.DataFrame(preprocessing.scale(data))
    testData = pd.DataFrame(preprocessing.scale(testData))

    # validation divides data into k parts, with 1 part set for validation
    trainData, trainAns, validData, validAns, validIds = getValidation(data, ans,
ids, k)

    return trainData, trainAns, validData, validAns, validIds, testData, testIds


def preprocess4(train, test, k):
    # select relevent columns
    data = train.loc[:,['price','badges_count','badge_product_quality',
'shipping_option_price','merchant_info_subtitle', 'merchant_rating',
'inventory_total']]
    # data = train.loc[:,
    #         ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_rating',
    #          'inventory_total']]
    # pull ids and ratings for saving to file
    ids = train.loc[:, 'id']
    ans = train.loc[:, 'rating']
    testData = test.loc[:,
           ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_info_subtitle',
            'merchant_rating', 'inventory_total']]

    # testData = test.loc[:,
    #             ['price', 'badges_count', 'badge_product_quality',
'shipping_option_price', 'merchant_rating',
    #              'inventory_total']]
    testIds = test.loc[:, 'id']

    # impute and encode
    cols = [i for i in data.columns]
    for i in cols:
        data[i] = pd.DataFrame(preprocessing.scale(impute_encode(data[i])))

    cols = [i for i in testData.columns]
    for i in cols:
        testData[i] = pd.DataFrame(preprocessing.scale(impute_encode(testData[i])))

    # does cross validation
    # returns trainData, trainAns, validationData, validationAns, validationIds
    trainData, trainAns, validData, validAns, validIds = getValidation(data, ans,
ids, k)

    return trainData, trainAns, validData, validAns, validIds, testData, testIds


# given k, divides data into k sections
def getValidation(data, ans, ids, k):
    hold = pd.concat([data,ans],axis=1)
    hold = hold.sample(frac=1)
    data = hold.iloc[:,:-1]
    ans = hold.iloc[:,-1]
    # size of each section - last section will be slightly bigger/smaller depending
on rounding
    length = len(data)
    size = round(length/k)
    # k section, each a 2D array of data
    trainData = list()
    trainAns = list()
    validationData = list()
    validationAns = list()
    validationIds = list()
    for i in range(k):
        # start and end point for validation
        start = i*size
        end = (i+1)*size

        trainData.append(data.iloc[0:start, :].append(data.iloc[end:,:],
ignore_index=True))
        trainAns.append(ans.iloc[0:start].append(ans.iloc[end:], ignore_index=True))

        # go to end for last one
        if i == k-1:
            end = length

        validationData.append(data.iloc[start:end, :])
        validationAns.append(ans.iloc[start:end])
        validationIds.append(ids.iloc[start:end])

    return trainData, trainAns, validationData, validationAns, validationIds


# trains NN based on data and answers
# predicts validation and test
# calculates accuracy of validation
def neuralNet(trainData, trainAns, validData, validAns, validIds, testData, testIds,
k, preprocess):
    # make NN, set parameters
    NN = MLPClassifier(solver='sgd', alpha=1e-3, hidden_layer_sizes=(120,30),
                       n_iter_no_change=20, max_iter=10000).fit(trainData, trainAns)
    # predict
    ans = NN.predict(testData)
    length = len(testIds)

    # save to file with ids
    toSave = np.zeros((length,2))
    for i in range(length):
        toSave[i][0] = testIds[i].astype(int)
        toSave[i][1] = (ans[i].astype(float)).round(decimals=1)

    #pd.DataFrame(toSave).to_csv("NNData/NNtest" + str(preprocess) + str(k) +".csv",
index=None, header=['id','rating'])
    np.savetxt("NNData/NNtest" + str(preprocess) + str(k) +".csv", toSave, fmt='%d,
%.1f', header='id,rating', comments='')

    # repeat for validation - gives approximate accuracy
    ans = NN.predict(validData)
    length = len(validData)
    toSave = np.zeros((length,3))
    count = length
    for i in range(length):
        if validAns.iloc[i] != ans[i]:
            count -= 1
        toSave[i][0] = validIds.iloc[i]
        toSave[i][1] = validAns.iloc[i]
        toSave[i][2] = ans[i]
    checkAns = NN.predict(trainData)
    # calculate score
    print(f1_score(validAns, ans, average='micro'), f1_score(checkAns, trainAns,
average='micro'))
    pd.DataFrame(toSave).to_csv("NNData/NNvalid" + str(preprocess) + str(k) +".csv",
index=None, header=['id','Actual','Predicted'])
    # np.savetxt("NNData/NNvalid" + str(preprocess) + str(k) + ".csv", toSave,
fmt='%d, %.1f', header='id,rating',
    #            comments='')


# trains RF based on data and answers, saves test and validation answers to file
# calculates percent/score
def randomForest(trainData, trainAns, validData, validAns, validIds, testData,
testIds, k, preprocess):
    # make RF, set parameters
    RF = RandomForestClassifier()
    RF.fit(trainData, trainAns)
    ans = RF.predict(testData)

    # save
    length = len(testIds)
    toSave = np.zeros((length,2))
    for i in range(length):
        toSave[i][0] = testIds[i]
        toSave[i][1] = ans[i]

    # pd.DataFrame(toSave).to_csv("RFData/RFtest" + str(preprocess) + str(k) +".csv",
index=None, header=['id','rating'])
    np.savetxt("RFData/RFtest" + str(preprocess) + str(k) + ".csv", toSave, fmt='%d,
%.1f', header='id,rating',
               comments='')
    # repeat for validation
    ans = RF.predict(validData)
    length = len(validData)
    toSave = np.zeros((length,3))
    count = length
    for i in range(length):
        if validAns.iloc[i] != ans[i]:
            count -= 1
        toSave[i][0] = validIds.iloc[i]
        toSave[i][1] = validAns.iloc[i]
        toSave[i][2] = ans[i]

    checkAns = RF.predict(trainData)

    print(f1_score(validAns, ans, average='micro'), f1_score(checkAns, trainAns,
average='micro'))
    pd.DataFrame(toSave).to_csv("RFData/RFvalid" + str(preprocess) + str(k) +".csv",
index=None, header=['id','Actual','Predicted'])
    # np.savetxt("RFData/RFvalid" + str(preprocess) + str(k) + ".csv", toSave,
fmt='%d, %.1f', header='id,rating',
    #            comments='')
# svm
def SVM(trainData, trainAns, validData, validAns, validIds, testData, testIds, k,
preprocess):
    # make svm
    # poly appears to be the best, based on validation scores
    clf = svm.SVC(kernel='poly', degree=3)
    clf.fit(trainData, trainAns)
    ans = clf.predict(testData)

    # save
    length = len(testIds)
    toSave = np.zeros((length, 2))
    for i in range(length):
        toSave[i][0] = testIds[i]
        toSave[i][1] = ans[i]

    # pd.DataFrame(toSave).to_csv("SVMData/SVMtest" + str(preprocess) + str(k)
+".csv", index=None, header=['id', 'rating'])
    np.savetxt("SVMData/SVMtest" + str(preprocess) + str(k) + ".csv", toSave,
fmt='%d, %.1f', header='id,rating',
               comments='')

    ans = clf.predict(validData)
    length = len(validData)
    toSave = np.zeros((length, 3))
    count = length
    for i in range(length):
        if validAns.iloc[i] != ans[i]:
            count -= 1
        toSave[i][0] = validIds.iloc[i]
        toSave[i][1] = validAns.iloc[i]
        toSave[i][2] = ans[i]

    checkAns = clf.predict(trainData)

    print(f1_score(validAns, ans, average='micro'),f1_score(checkAns, trainAns,
average='micro'))
    pd.DataFrame(toSave).to_csv("SVMData/SVMvalid" + str(preprocess) + str(k) +
".csv", index=None,
                                header=['id', 'Actual', 'Predicted'])
    # np.savetxt("SVMData/SVMvalid" + str(preprocess) + str(k) + ".csv", toSave,
fmt='%d, %.1f', header='id,rating',
    #            comments = '')

def main(trainFile, testFile):
    # datafiles
    train = pd.read_csv(trainFile)
    test = pd.read_csv(testFile)

    # for correlation
    #heatmap(train)

    k = 5

    # using preprocess 1 - has shown best results so far
    # random forest has highest score
    trainData, trainAns, validData, validAns, validIds, testData, testIds = 
preprocess1(train, test, k)
    preprocess = 1
    for i in range(k):
        #neuralNet(trainData[i], trainAns[i], validData[i], validAns[i], validIds[i], 
testData, testIds, i, preprocess)
        randomForest(trainData[i], trainAns[i], validData[i], validAns[i], 
validIds[i], testData, testIds, i, preprocess)
        #SVM(trainData[i], trainAns[i], validData[i], validAns[i], validIds[i], 
testData, testIds, i, preprocess)
        print()
    print()

    # using preprocess 2
    # trainData, trainAns, validData, validAns, validIds, testData, testIds = 
preprocess2(train, test, k)
    # preprocess = 2
    # for i in range(k):
    #     neuralNet(trainData[i], trainAns[i], validData[i], validAns[i], 
validIds[i], testData, testIds, i, preprocess)
    #     randomForest(trainData[i], trainAns[i], validData[i], validAns[i], 
validIds[i], testData, testIds, i, preprocess)
    #     SVM(trainData[i], trainAns[i], validData[i], validAns[i], validIds[i], 
testData, testIds, i, preprocess)
    #     print()
    # print()
    #
    # # using preprocess 3
    # trainData, trainAns, validData, validAns, validIds, testData, testIds = 
preprocess3(train, test, k)
    # preprocess = 3
    # for i in range(k):
    #     neuralNet(trainData[i], trainAns[i], validData[i], validAns[i], 
validIds[i], testData, testIds, i, preprocess)
    #     randomForest(trainData[i], trainAns[i], validData[i], validAns[i], 
validIds[i], testData, testIds, i, preprocess)
    #     SVM(trainData[i], trainAns[i], validData[i], validAns[i], validIds[i], 
testData, testIds, i, preprocess)
    #     print()


trainFile = 'train_new.csv'
#train = pd.read_csv(trainFile)
#heatmap(train)
testFile = 'test_new.csv'
main(trainFile, testFile)
```