A Comparison of Supervised Learning Algorithms on Binary Classification Accuracy

Enoch Li
Department of Cognitive Science
University of California San Diego
La Jolla, CA 92122 USA
e2li@ucsd.edu

Abstract

Previous research of supervised machine learning algorithms have looked at a comprehension comparison of these models on multiple performance metrics. This report presents a comparison between three supervised machine learning models: k nearest neighbors, logistic regression, and random forests. An accuracy score is used for evaluating each learning model. It is important to note that this paper attempts to replicate a portion of a previously published analysis.

All source code for this paper can be found on GitHub.

1 Introduction

Machine learning algorithms are widely used both in academia and industry, with libraries such as `scikit-learn` [4] optimizing the implementation of these algorithms while also providing tools for model fitting, data preprocessing, and evaluation. While these algorithms have become easier to use, it is still important to evaluate the performance made by these classifiers, as these libraries do not provide the performance metrics on how optimal or sub-optimal a classifier may be.

The goal of this analysis is to evaluate the accuracy performance on three binary classification problems using three supervised machine learning algorithms: K Nearest Neighbors, Logistic Regression, and Random Forests. Accuracy is measured as the number of correctly predicted entries in the testing set. Each algorithm was implemented using the `scikit-learn` library along with helper tools to evaluate the cross-validation and scoring metrics. The intent of this paper is to perform similar analysis done in "An Empirical Comparison of Supervised Learning Algorithms" [1]. Thus, the methodology of this paper will closely resemble that of Caruana, R., & Niculescu-Mizil, A. (2006), henceforth referred to as "Caruana".

2 Methods

Caruana presents a comparison of ten supervised machine learning algorithms on eight performance metrics. However, this paper will only detail the implementation and parameters used for each of the three algorithms on the accuracy performance metric.

2.1 K Nearest Neighbors (KNN)

KNN was implemented using `KNeighborsClassifier` from the `scikit-learn` library using distance weighting. Caruana used 26 values of K ranging from K = 1 to K = |*trainset*|. For this report, only 25 values of K were used, evenly log-spaced ranging from K = 1 to K = 500. All other parameters for **KNN** were set to the default values provided by `KNeighborsClassifier`.

2.2 Logistic Regression (LOGIT)

LOGIT was implemented using `LogisticRegression` from the `scikit-learn` library using both unregularized and regularized models. Unregularized models had no penalty, while the regularized models used l1 and l2 penalties. Similar to Caruana, The regularization parameters were varied by factors of 10 from 10^{-8} to 10^{4}, including regularization parameter = 0, yielding 14 total parameter settings. The `'newton-cg'` and `'saga'` solvers were also used. All other parameters for **LOGIT** were set to the default values provided by `LogisticRegression`.

2.3 Random Forests (RF)

RF was implemented using `RandomForestClassifier` from the `scikit-learn` library using a total of 1024 trees. Similar to Caruana, the size of the feature set for each split are as follows: 1, 2, 4, 6, 8, 12, 16, and 20. All other parameters for **RF** were set to the default values provided by `RandomForestClassifier`.

3 Data Sets

Each of the algorithms were compared on three binary classification problems using data sets from the UCI Repository [2]. Because the data sets contain categorical information, they have been converted to continuous values by transforming each unique attribute into binary values via one-hot encoding (one binary value per unique attribute). The methods for conversion to binary problems are found in Caruana.

3.1 Adult

The **Adult** data set consists of 32561 entries and 14 attributes, with 2399 of these entries containing missing values. Of these 14 attributes, 8 contained categorical values. After transforming the **Adult** data set to a continuous attributes and dropping the missing entries, it consists of 30162 entries and 104 attributes. **Adult** was converted to a binary problem by treating income values $\leq$50K as negative and >50K as positive.

3.2 Covertype (Cover)

The **Cover** data set consists of 581012 entries and 54 attributes, with 0 of these entries containing missing values. Of these 54 attributes, 0 contained categorical values. **Cover** was converted to a binary problem by treating the most common cover type as positive and the remainder as negative.

3.3 Letter Recognition (Letter)

The **Letter** data set consists of 20000 entries and 16 attributes, with 0 of these entries containing missing values. Of these 16 attributes, 0 contained categorical values. **Letter** was converted to a binary problem in two ways, unbalanced (**Letter**$_{p1}$) and balanced (**Letter**$_{p2}$). **Letter**$_{p1}$ treats the letter "O" as positive and the remainder of the letters as negative, while **Letter**$_{p2}$ takes letters "A-M" as positive and the rest of the letters as negative.

4 Experiment

For each algorithm, a total of 3 trials were run on each data set. For each trial, 5000 samples were randomly selected as the training set with the remainder set aside as a testing set. Using `GridSearchCV` from the `scikit-learn` library, a 5-fold cross validation was used on the training set with the specifications mentioned in Section 2. Each algorithm is trained on 4000 samples (4 folds) and evaluated on the final 1000 samples (1 fold) to determine the best parameters. The best classifier is trained again using the entire training data before being tested using the testing set.

Since this paper is focused on accuracy, only the accuracy performance metric will be reported for each classifier. That is, the number of correctly predicted entries out of the total entries in a given set. For this analysis, `accuracy_score` from the `scikit-learn` library was used to calculate accuracy.

Following the reporting from Caruana, the algorithm with the best accuracy performance has been **bolded**. Algorithms with performances that are not significantly different to the best score using an uncorrected two sample t-tests with p = 0.05 are annotated with an * symbol. Algorithm performances that are unmarked indicate an accuracy score that is *significantly lower* than the best algorithm performance.

Table 1: Mean testing accuracy score between algorithms by data set

Algorithm	Adult	Cover	Letter$_{p1}$	Letter$_{p2}$
KNN	0.8258	0.7807	**0.9911**	**0.9568**
LOGIT	0.8453*	0.7567	0.9622	0.7265
RF	**0.8460**	**0.8215**	0.9874*	0.9455

Table 1 describes a comparison of mean testing accuracy scores between algorithms for each data set. (See Table A for raw accuracy scores.) The p-values between algorithms by data set are found in Table B.

Table 2: Mean testing accuracy score between algorithms

Algorithm	Accuracy
KNN	0.8886*
LOGIT	0.8227
RF	**0.9001**

Table 2 describes a comparison of mean testing accuracy scores between algorithms. The p-values between algorithms are found in Table C.

Table 3: Mean training accuracy score between algorithms by data set

Algorithm	Adult	Cover	Letter$_{p1}$	Letter$_{p2}$
KNN	1.0000	1.0000	1.0000	1.0000
LOGIT	0.8498	0.7611	0.9628	0.7309
RF	1.0000	1.0000	1.0000	1.0000

Table 3 describes a comparison of mean training accuracy scores between algorithms for each data set.

5 Results

From Table 1, it appears **KNN** performs the best on both the **Letter**$_{p1}$ and **Letter**$_{p2}$ data sets, while **RF** performed the best on the **Adult** and **Cover** data sets. On the other hand, **LOGIT** did not perform the best for any of the data sets.

Because there are multiple algorithms that performed the best, there does not appear to be a best algorithm over all of the data sets. This conclusion is reflected in Table 2 since, although **RF** had the best mean accuracy score, **KNN** 's accuracy score did not differ significantly. These results are also consistent with the results from Caruana, both for accuracy scores over each data set and overall accuracy scores.

One metric of importance is the accuracy comparison for each of these algorithms on unbalanced (**Letter**$_{p1}$) data versus balanced (**Letter**$_{p2}$) data. Table 1 shows that all three algorithms have excellent

accuracy with the unbalanced data. However, the accuracy scores for all three algorithms decrease when comparing on balanced data, especially for **LOGIT** .

A point of discussion to note is the differences between the training and testing accuracy performance for each of the algorithms used. Table 3 shows that both **KNN** and **RF** had an accuracy score of 1 over all the data sets. This is an indication that these algorithms are most likely overfitting the data they were trained on, which may have lead to the higher accuracy performances on the testing data that were recorded. **KNN** and **RF** are known to overfit data during training, but future analysis could be beneficial to determine how influencial overfitting the training data has on testing performance of these algorithms.

It is also interesting to note that the **LOGIT** training and testing accuracy performances were nearly identical (less than 0.5% difference). This suggests that the testing data set was a representation of the training set. A future analysis could do more trials using **LOGIT** to see if these results are due to random noise or if the data representation does not affect **LOGIT** heavily.

6 Conclusion

It is clear that the applications of machine learning are endless and the field continues to grow and become easier to use. The simplicity of running and implementing these machine learning models was astounding, producing results that were comparable to other research results. This paper demonstrated that both **RF** and **KNN** had the best performance, with **RF** doing insignificantly better, at the cost of overfitting. And while **LOGIT** performed the worst, it was apparent that even poor models can perform well in certain areas. With more computational resources and time, this paper could be extended to explore the performance of other supervised machine learning models, as well as understanding in what situations would one model would perform better than another.

7 Bonus Points

I believe that my work on this paper deserves bonus points due the additional work done by splitting the analysis of the **Letter** data set into balanced and unbalanced data, thus performing 36 trials instead of the required 27. I investigated the comparison of the accuracy results between balanced and unbalanced data and found interesting results, which are included in my report in Section 5.

References

[1] Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning* (pp. 161-168).

[2] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Fleischer, J. (2020). COGS 118A. Supervised Machine Learning Algorithms. University of California San Diego, La Jolla, CA.

[4] Scikit-learn: Machine Learning in Python Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Appendix A

Table A: Raw test set scores between algorithms by data set

Data Set (Trial #)	KNN	LOGIT	RF
Adult 1	0.8299	0.8461	0.8478
Adult 2	0.8224	0.8454	0.8441
Adult 3	0.8251	0.8445	0.8461
Cover 1	0.7835	0.7548	0.8213
Cover 2	0.7766	0.7540	0.8168
Cover 3	0.7819	0.7613	0.8264
Letter$_{p1}$ 1	0.9911	0.9613	0.9853
Letter$_{p1}$ 2	0.9913	0.9625	0.9893
Letter$_{p1}$ 3	0.9909	0.9629	0.9877
Letter$_{p2}$ 1	0.9553	0.7265	0.9436
Letter$_{p2}$ 2	0.9567	0.7259	0.9499
Letter$_{p2}$ 3	0.9583	0.7270	0.9431

Table B: P-value between algorithms by data set

Algorithms	Adult	Cover	Letter$_{p1}$	Letter$_{p2}$
KNN / LOGIT	0.0100	0.0102	0.0003	0.0000
LOGIT /RF	0.5679	0.0003	0.0011	0.0001
RF / KNN	0.0033	0.0023	0.0822	0.0440

Table C: P-value between algorithms

Algorithms	P-value
KNN / LOGIT	0.0452
LOGIT / RF	0.0116
RF / KNN	0.0911

Code

The attached code is organized in the following order:

Data Cleaning

December 16, 2020

1 Data Cleaning

This notebook is to be used for data cleaning and preparing the data sets for selecting hyper-paramters of sklearn models. The classification label will be listed as the last column

[1]:
```
# import necessary packages and declare global variables
import pandas as pd
import numpy as np
import string
bold = '\033[1m'
unbold = '\033[0m'
```

[2]:
```
# import datasets and assign headers
adult_column = ["age", "workclass", "num_represented", "education",␣
 ↪"education_num", "marital_status", "occupation",
                "relationship", "race", "sex", "capital_gain", "capital_loss",␣
 ↪"hours_per_week", "country", "income"]

cover_column = ["elevation", "aspect", "slope", "h_dist_to_water",␣
 ↪"v_dist_to_water", "h_dist_to_road", "shade_am",
                "shade_noon", "shade_pm", "h_dist_to_fire", "wilderness_area1",␣
 ↪"wilderness_area2", "wilderness_area3",
                "wilderness_area4", 'soil_type1', 'soil_type2', 'soil_type3',␣
 ↪'soil_type4', 'soil_type5', 'soil_type6',
                'soil_type7', 'soil_type8', 'soil_type9', 'soil_type10',␣
 ↪'soil_type11', 'soil_type12', 'soil_type13',
                'soil_type14', 'soil_type15', 'soil_type16', 'soil_type17',␣
 ↪'soil_type18', 'soil_type19', 'soil_type20',
                'soil_type21', 'soil_type22', 'soil_type23',␣
 ↪'soil_type24','soil_type25', 'soil_type26', 'soil_type27',
                'soil_type28', 'soil_type29', 'soil_type30', 'soil_type31',␣
 ↪'soil_type32', 'soil_type33', 'soil_type34',
                'soil_type35', 'soil_type36', 'soil_type37', 'soil_type38',␣
 ↪'soil_type39', 'soil_type40',  "cover_type"]

letter_column = ["letter", "x_box", "y_box", "width", "height", "total_pix",␣
 ↪"x_bar", "y_bar", "x2_bar",
```

```
                "y2_bar", "xy_bar", "x2y_bar", "xy2_bar", "x_edge",␣
↪"x_edge_y", "y_edge", "y_edge_x"]

adult = pd.read_csv('Original Data/adult.csv', names=adult_column, na_values='?
↪')
cover = pd.read_csv('Original Data/covtype.csv', names=cover_column,␣
↪na_values='?')
letter = pd.read_csv('Original Data/letter-recognition.csv',␣
↪names=letter_column, na_values='?')
```

1.1 Describing the datasets

[3]:
```
# describe the adult dataset
print(bold+"Adult dataset"+unbold)
print("{} rows, {} attributes".format(adult.shape[0], adult.shape[1]))
print("{} missing values".format(adult.isnull().sum().sum()))
print("{} rows with missing values".format(adult.isnull().any(axis=1).sum()))
adult.head()
```

Adult dataset
32561 rows, 15 attributes
4262 missing values
2399 rows with missing values

[3]:

	age	workclass	num_represented	education	education_num	\
0	39	State-gov	77516	Bachelors	13	
1	50	Self-emp-not-inc	83311	Bachelors	13	
2	38	Private	215646	HS-grad	9	
3	53	Private	234721	11th	7	
4	28	Private	338409	Bachelors	13	

	marital_status	occupation	relationship	race	sex	\
0	Never-married	Adm-clerical	Not-in-family	White	Male	
1	Married-civ-spouse	Exec-managerial	Husband	White	Male	
2	Divorced	Handlers-cleaners	Not-in-family	White	Male	
3	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	
4	Married-civ-spouse	Prof-specialty	Wife	Black	Female	

	capital_gain	capital_loss	hours_per_week	country	income
0	2174	0	40	United-States	<=50K
1	0	0	13	United-States	<=50K
2	0	0	40	United-States	<=50K
3	0	0	40	United-States	<=50K
4	0	0	40	Cuba	<=50K

[4]:
```
# descibe the cover dataset
print(bold+"Cover dataset"+unbold)
print("{} rows, {} attributes".format(cover.shape[0], cover.shape[1]))
print("{} missing values".format(cover.isnull().sum().sum()))
cover.head()
```

Cover dataset
581012 rows, 55 attributes
0 missing values

[4]:

	elevation	aspect	slope	h_dist_to_water	v_dist_to_water	h_dist_to_road	\
0	2596	51	3	258	0	510	
1	2590	56	2	212	-6	390	
2	2804	139	9	268	65	3180	
3	2785	155	18	242	118	3090	
4	2595	45	2	153	-1	391	

	shade_am	shade_noon	shade_pm	h_dist_to_fire	…	soil_type32	\
0	221	232	148	6279	…	0	
1	220	235	151	6225	…	0	
2	234	238	135	6121	…	0	
3	238	238	122	6211	…	0	
4	220	234	150	6172	…	0	

	soil_type33	soil_type34	soil_type35	soil_type36	soil_type37	\
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	0	0	
3	0	0	0	0	0	
4	0	0	0	0	0	

	soil_type38	soil_type39	soil_type40	cover_type
0	0	0	0	5
1	0	0	0	5
2	0	0	0	2
3	0	0	0	2
4	0	0	0	5

[5 rows x 55 columns]

[5]:
```
# display how many missing values are in each dataset
print(bold+"Letter dataset"+unbold)
print("{} rows, {} attributes".format(letter.shape[0], letter.shape[1]))
print("{} missing values".format(letter.isnull().sum().sum()))
letter.head()
```

Letter dataset
20000 rows, 17 attributes

0 missing values

[5]:

	letter	x_box	y_box	width	height	total_pix	x_bar	y_bar	x2_bar
0	T	2	8	3	5	1	8	13	0
1	I	5	12	3	7	2	10	5	5
2	D	4	11	6	8	6	10	6	2
3	N	7	11	6	6	3	5	9	4
4	G	2	1	3	1	1	8	6	6

	y2_bar	xy_bar	x2y_bar	xy2_bar	x_edge	x_edge_y	y_edge	y_edge_x
0	6	6	10	8	0	8	0	8
1	4	13	3	9	2	8	4	10
2	6	10	3	7	3	7	3	9
3	6	4	4	10	6	10	2	8
4	6	6	5	9	1	7	5	10

1.2 Cleaning Adult Dataset

The Adult dataset has missing values and categorical data, so we need to remove the rows with missing data and convert all the categorical data into numerical data.

[6]:
```
# show breakdown of missing values
print(bold+"Breakdown of missing values in Adult dataset"+unbold)
print(adult.isnull().sum())

# drop the rows with na in the adult dataset
adult.dropna(inplace=True)
print('\n'+bold+"Shape of adult with dropped rows"+unbold)
print("{} rows, {} attributes".format(adult.shape[0], adult.shape[1]))
```

Breakdown of missing values in Adult dataset

```
age                  0
workclass         1836
num_represented      0
education            0
education_num        0
marital_status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital_gain         0
capital_loss         0
hours_per_week       0
country            583
income               0
dtype: int64
```

Shape of adult with dropped rows
30162 rows, 15 attributes

[7]:
```
# transform categorical data into binary values
left = pd.get_dummies(adult[adult.columns[:len(adult.columns)-1]])

# income <=50K is rep. as -1, >50K is rep. as 1
right = adult['income'].replace(to_replace=['<=50K','>50K'], value=[-1,1])

# adult dataset is now ready for modeling
adult_clean = pd.concat([left,right],axis=1)

# final shape of clean dataset attributes
print("{} rows, {} attributes".format(adult_clean.iloc[:,:-1].shape[0],␣
 ↪adult_clean.iloc[:,:-1].shape[1]))

# # export dataset to new csv file
adult_clean.to_csv('adult_clean.csv',index=False)
```

30162 rows, 104 attributes

1.3 Cleaning Cover Dataset

The Cover dataset is converted to numerical data by treating the most common cover type as the positive and the rest as negative.

[8]:
```
# Calculate the binary data for cover type
largest = cover['cover_type'].value_counts().idxmax()
rest = list(filter(lambda c: c != largest, cover['cover_type'].unique()))
negCov = [-1] * len(rest)

# transform the cover_type data into binary values
cover_type = cover['cover_type'].replace(to_replace=[largest, *rest], value=[1,␣
 ↪*negCov])

# cover dataset is now ready for modeling
cover_clean = pd.concat([cover[cover.columns[:len(cover.
 ↪columns)-1]],cover_type],axis=1)

# final shape of clean dataset attributes
print("{} rows, {} attributes".format(cover.iloc[:,:-1].shape[0], cover.iloc[:,:
 ↪-1].shape[1]))

# export dataset to new csv file
cover_clean.to_csv('cover_clean.csv',index=False)
```

581012 rows, 54 attributes

1.4 Cleaning Letter Dataset

The Letter dataset is will be converted to numerical data in two different ways.

letter_p1 treats "O" as positive and the remaining 25 letters as negative, yielding a very unbalanced problem. letter_p2 uses letters A-M as positives and the rest as negatives, yielding a well balanced problem.

[9]:
```
# separate out the letter column
let = letter['letter']

# get the alphabet
alpha = list(string.ascii_uppercase)
```

Getting letter_p1 dataset

[10]:
```
# separate "O" from the rest of the alphabet
other = list(filter(lambda a: a != "O", alpha))
neg_p1 = [-1] * len(other)

# transform the letter column into the p1 numerical values
p1 = let.replace(to_replace=["O", *other], value=[1,*neg_p1])

# letter_p1 dataset is now ready for modeling
letter_p1 = pd.concat([letter[letter.columns[1:]],p1],axis=1)

# final shape of clean dataset attributes
print("{} rows, {} attributes".format(letter_p1.iloc[:,:-1].shape[0], letter_p1.
 ↪iloc[:,:-1].shape[1]))

# export dataset to new csv file
letter_p1.to_csv('letter_p1.csv',index=False)
```

```
20000 rows, 16 attributes
```

Getting letter_p2 dataset

[11]:
```
# separate the two halfs of the alphabet
upper = alpha[:len(alpha)//2]
lower = alpha[len(alpha)//2:]
pos_p2 = [1] * len(upper)
neg_p2 = [-1] * len(lower)

# transform the letter column into the p1 numerical values
p2 = let.replace(to_replace=[*upper, *lower], value=[*pos_p2,*neg_p2])

# letter_p2 dataset is now ready for modeling
letter_p2 = pd.concat([letter[letter.columns[1:]],p2],axis=1)
```

```
# final shape of clean dataset attributes
print("{} rows, {} attributes".format(letter_p2.iloc[:,:-1].shape[0], letter_p2.
 ↪iloc[:,:-1].shape[1]))

# export dataset to new csv file
letter_p2.to_csv('letter_p2.csv',index=False)
```

```
20000 rows, 16 attributes
```

K Nearest Neighbors Modeling

December 16, 2020

0.1 K Nearest Neighbors

For K Nearest Neighbors, this is the outline of what I will be doing

[1]:
```
# MAKE SURE TO RUN THIS CELL BEFORE MAKING EDITS

# import libaries and set global variables
import os.path
import pandas as pd
import numpy as np
import joblib
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
```

[2]:
```
# read in the datasets for kNN
adult = pd.read_csv('../Data/adult_clean.csv')
cover = pd.read_csv('../Data/cover_clean.csv')
letter_p1 = pd.read_csv('../Data/letter_p1.csv')
letter_p2 = pd.read_csv('../Data/letter_p2.csv')
```

[3]:
```
# separate the datasets into the data (X) and labels (Y)
adult_X = adult.iloc[:,:-1]
adult_Y = adult.iloc[:,-1]

cover_X = cover.iloc[:,:-1]
cover_Y = cover.iloc[:,-1]

letter_p1_X = letter_p1.iloc[:,:-1]
letter_p1_Y = letter_p1.iloc[:,-1]

letter_p2_X = letter_p2.iloc[:,:-1]
letter_p2_Y = letter_p2.iloc[:,-1]
```

[4]:
```
# get k values
k = list(np.logspace(0,2.699, 25,dtype='int'))  # 2.699 is the approx. equiv.␣
 ↪to log(500)/log(10)

# define the cross-validation
cv = StratifiedKFold(n_splits=5)

# build the pipeline
pipe = Pipeline([('std', StandardScaler()),
                 ('classifier', KNeighborsClassifier())])

# setting up the parameters
param = {'classifier__n_neighbors': k,
         'classifier__weights': ['distance']}

# create grid search
clf = GridSearchCV(estimator=pipe, param_grid=param, cv=cv, n_jobs=-1,␣
 ↪verbose=0)
```

0.1.1 Run KNN on Adult dataset for 3 trials

[5]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_adult_results.txt"):
    os.remove("Results/best_adult_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for testing
    adult_X_train, adult_X_test, adult_Y_train, adult_Y_test =␣
 ↪train_test_split(adult_X,adult_Y,
                                                                        ␣
 ↪train_size=5000, random_state=trial)

    # fit the grid search on adult dataset and save the model
    adult_model = clf.fit(adult_X_train, adult_Y_train)
    joblib.dump(adult_model, 'Models/adult_model_'+str(trial)+'.pkl',␣
 ↪compress=1)

    # fit best parameters to training set and save the model
    best_adult_model = adult_model.fit(adult_X_train, adult_Y_train)
    joblib.dump(best_adult_model, 'Models/best_adult_model_'+str(trial)+'.pkl',␣
 ↪compress=1)
```

```
    # get the accuracy score for the testing and training data
    train_acc = accuracy_score(y_true=adult_Y_train, y_pred=best_adult_model.
↪predict(adult_X_train))
    test_acc = accuracy_score(y_true=adult_Y_test, y_pred=best_adult_model.
↪predict(adult_X_test))

    # write the accuracy score into a file
    f = open("Results/best_adult_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_adult_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_adult_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 1min 26s, sys: 1.67 s, total: 1min 28s
Wall time: 2min 40s
```

0.1.2 Run KNN on Cover dataset for 3 trials

[6]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_cover_results.txt"):
    os.remove("Results/best_cover_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)    # print the trial number

    # split the data into 5000 points for training and save the rest for testing
    cover_X_train, cover_X_test, cover_Y_train, cover_Y_test =␣
↪train_test_split(cover_X,cover_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on cover dataset and save the model
    cover_model = clf.fit(cover_X_train, cover_Y_train)
    joblib.dump(cover_model, 'Models/cover_model_'+str(trial)+'.pkl',␣
↪compress=1)

    # fit best parameters to training set and save the model
```

```
    best_cover_model = cover_model.fit(cover_X_train, cover_Y_train)
    joblib.dump(best_cover_model, 'Models/best_cover_model_'+str(trial)+'.pkl',␣
↪compress=1)

    train_acc = accuracy_score(y_true=cover_Y_train, y_pred=best_cover_model.
↪predict(cover_X_train))
    test_acc = accuracy_score(y_true=cover_Y_test, y_pred=best_cover_model.
↪predict(cover_X_test))

    f = open("Results/best_cover_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_cover_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_cover_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 10min 30s, sys: 7 s, total: 10min 37s
Wall time: 11min 15s
```

0.1.3 Run KNN on Letter_p1 dataset for 3 trials

[7]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_letter_p1_results.txt"):
    os.remove("Results/best_letter_p1_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for␣
↪testing
    letter_p1_X_train, letter_p1_X_test, letter_p1_Y_train, letter_p1_Y_test =␣
↪train_test_split(letter_p1_X,letter_p1_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on letter_p1 dataset and save the model
    letter_p1_model = clf.fit(letter_p1_X_train, letter_p1_Y_train)
    joblib.dump(letter_p1_model, 'Models/letter_p1_model_'+str(trial)+'.pkl',␣
↪compress=1)
```

```
    # fit best parameters to training set and save the model
    best_letter_p1_model = letter_p1_model.fit(letter_p1_X_train,␣
↪letter_p1_Y_train)
    joblib.dump(best_letter_p1_model, 'Models/
↪best_letter_p1_model_'+str(trial)+'.pkl', compress=1)

    train_acc = accuracy_score(y_true=letter_p1_Y_train,␣
↪y_pred=best_letter_p1_model.predict(letter_p1_X_train))
    test_acc = accuracy_score(y_true=letter_p1_Y_test,␣
↪y_pred=best_letter_p1_model.predict(letter_p1_X_test))

    f = open("Results/best_letter_p1_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_letter_p1_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_letter_p1_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 6.66 s, sys: 290 ms, total: 6.95 s
Wall time: 19.7 s
```

0.1.4 Run KNN on Letter_p2 dataset for 3 trials

[8]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_letter_p2_results.txt"):
    os.remove("Results/best_letter_p2_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for testing
    letter_p2_X_train, letter_p2_X_test, letter_p2_Y_train, letter_p2_Y_test =␣
↪train_test_split(letter_p2_X,letter_p2_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on letter_p2 dataset and save the model
    letter_p2_model = clf.fit(letter_p2_X_train, letter_p2_Y_train)
```

```
   joblib.dump(letter_p2_model, 'Models/letter_p2_model_'+str(trial)+'.pkl',␣
↪compress=1)

   # fit best parameters to training set and save the model
   best_letter_p2_model = letter_p2_model.fit(letter_p2_X_train,␣
↪letter_p2_Y_train)
   joblib.dump(best_letter_p2_model, 'Models/
↪best_letter_p2_model_'+str(trial)+'.pkl', compress=1)

   train_acc = accuracy_score(y_true=letter_p2_Y_train,␣
↪y_pred=best_letter_p2_model.predict(letter_p2_X_train))
   test_acc = accuracy_score(y_true=letter_p2_Y_test,␣
↪y_pred=best_letter_p2_model.predict(letter_p2_X_test))

   f = open("Results/best_letter_p2_results.txt", "a")
   f.write('Trial '+str(trial)+":\n")
   f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_letter_p2_model.best_score_))
   f.write('\tBest Parameters: %s\n' % best_letter_p2_model.best_params_)
   f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
   f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
   f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 7.35 s, sys: 269 ms, total: 7.61 s
Wall time: 21.2 s
```

Logistic Regression Modeling

December 16, 2020

0.1 Logistic Regression

For Logistic Regression, this is the outline of what I will be doing

[1]:
```
# MAKE SURE TO RUN THIS CELL BEFORE MAKING EDITS

# import libaries and set global variables
import os.path
import pandas as pd
import numpy as np
import joblib
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
```

[2]:
```
# read in the datasets for logistic regression
adult = pd.read_csv('../Data/adult_clean.csv')
cover = pd.read_csv('../Data/cover_clean.csv')
letter_p1 = pd.read_csv('../Data/letter_p1.csv')
letter_p2 = pd.read_csv('../Data/letter_p2.csv')
```

[3]:
```
# separate the datasets into the data (X) and labels (Y)
adult_X = adult.iloc[:,:-1]
adult_Y = adult.iloc[:,-1]

cover_X = cover.iloc[:,:-1]
cover_Y = cover.iloc[:,-1]

letter_p1_X = letter_p1.iloc[:,:-1]
letter_p1_Y = letter_p1.iloc[:,-1]

letter_p2_X = letter_p2.iloc[:,:-1]
letter_p2_Y = letter_p2.iloc[:,-1]
```

[4]:
```
# get regularization parameter values
reg = [0]+list(np.geomspace(10**-8,10**4, 13))

# define the cross-validation
cv = StratifiedKFold(n_splits=5)

# build the pipeline
pipe = Pipeline([('std', StandardScaler()),
                 ('classifier', LogisticRegression())])

# setting up the parameters
param = {'classifier__solver': ['saga', 'newton-cg'],
         'classifier__penalty': ['none', 'l1', 'l2'],
         'classifier__C': reg,
         'classifier__max_iter': [4000]}

# # create grid search
clf = GridSearchCV(estimator=pipe, param_grid=param, cv=cv, n_jobs=-1,␣
↪verbose=0)
```

0.1.1 Run Logit on Adult dataset for 3 trials

[5]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_adult_results.txt"):
    os.remove("Results/best_adult_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for testing
    adult_X_train, adult_X_test, adult_Y_train, adult_Y_test =␣
↪train_test_split(adult_X,adult_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on adult dataset and save the model
    adult_model = clf.fit(adult_X_train, adult_Y_train)
    joblib.dump(adult_model, 'Models/adult_model_'+str(trial)+'.pkl',␣
↪compress=1)

    # fit best parameters to training set and save the model
    best_adult_model = adult_model.fit(adult_X_train, adult_Y_train)
```

```
    joblib.dump(best_adult_model, 'Models/best_adult_model_'+str(trial)+'.pkl',␣
↪compress=1)

    # get the accuracy score for the testing and training data
    train_acc = accuracy_score(y_true=adult_Y_train, y_pred=best_adult_model.
↪predict(adult_X_train))
    test_acc = accuracy_score(y_true=adult_Y_test, y_pred=best_adult_model.
↪predict(adult_X_test))

    # write the accuracy score into a file
    f = open("Results/best_adult_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_adult_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_adult_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 40.7 s, sys: 9.66 s, total: 50.3 s
Wall time: 7min 3s
```

0.1.2 Run Logit on Cover dataset for 3 trials

[6]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_cover_results.txt"):
    os.remove("Results/best_cover_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)    # print the trial number

    # split the data into 5000 points for training and save the rest for testing
    cover_X_train, cover_X_test, cover_Y_train, cover_Y_test =␣
↪train_test_split(cover_X,cover_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on cover dataset and save the model
    cover_model = clf.fit(cover_X_train, cover_Y_train)
    joblib.dump(cover_model, 'Models/cover_model_'+str(trial)+'.pkl',␣
↪compress=1)
```

```
    # fit best parameters to training set and save the model
    best_cover_model = cover_model.fit(cover_X_train, cover_Y_train)
    joblib.dump(best_cover_model, 'Models/best_cover_model_'+str(trial)+'.pkl',␣
↪compress=1)

    train_acc = accuracy_score(y_true=cover_Y_train, y_pred=best_cover_model.
↪predict(cover_X_train))
    test_acc = accuracy_score(y_true=cover_Y_test, y_pred=best_cover_model.
↪predict(cover_X_test))

    f = open("Results/best_cover_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_cover_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_cover_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 28 s, sys: 1min 35s, total: 2min 3s
Wall time: 7min 1s
```

0.1.3 Run Logit on Letter_p1 dataset for 3 trials

[7]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_letter_p1_results.txt"):
    os.remove("Results/best_letter_p1_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for␣
↪testing
    letter_p1_X_train, letter_p1_X_test, letter_p1_Y_train, letter_p1_Y_test =␣
↪train_test_split(letter_p1_X,letter_p1_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on letter_p1 dataset and save the model
    letter_p1_model = clf.fit(letter_p1_X_train, letter_p1_Y_train)
```

```
    joblib.dump(letter_p1_model, 'Models/letter_p1_model_'+str(trial)+'.pkl',␣
↪compress=1)

    # fit best parameters to training set and save the model
    best_letter_p1_model = letter_p1_model.fit(letter_p1_X_train,␣
↪letter_p1_Y_train)
    joblib.dump(best_letter_p1_model, 'Models/
↪best_letter_p1_model_'+str(trial)+'.pkl', compress=1)

    train_acc = accuracy_score(y_true=letter_p1_Y_train,␣
↪y_pred=best_letter_p1_model.predict(letter_p1_X_train))
    test_acc = accuracy_score(y_true=letter_p1_Y_test,␣
↪y_pred=best_letter_p1_model.predict(letter_p1_X_test))

    f = open("Results/best_letter_p1_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_letter_p1_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_letter_p1_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
CPU times: user 6.43 s, sys: 8.15 s, total: 14.6 s
Wall time: 27 s
```

0.1.4 Run Logit on Letter_p2 datasets for 3 trials

[8]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_letter_p2_results.txt"):
    os.remove("Results/best_letter_p2_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for testing
    letter_p2_X_train, letter_p2_X_test, letter_p2_Y_train, letter_p2_Y_test =␣
↪train_test_split(letter_p2_X,letter_p2_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)
```

```
    # fit the grid search on letter_p2 dataset and save the model
    letter_p2_model = clf.fit(letter_p2_X_train, letter_p2_Y_train)
    joblib.dump(letter_p2_model, 'Models/letter_p2_model_'+str(trial)+'.pkl',␣
↪compress=1)

    # fit best parameters to training set and save the model
    best_letter_p2_model = letter_p2_model.fit(letter_p2_X_train,␣
↪letter_p2_Y_train)
    joblib.dump(best_letter_p2_model, 'Models/
↪best_letter_p2_model_'+str(trial)+'.pkl', compress=1)

    train_acc = accuracy_score(y_true=letter_p2_Y_train,␣
↪y_pred=best_letter_p2_model.predict(letter_p2_X_train))
    test_acc = accuracy_score(y_true=letter_p2_Y_test,␣
↪y_pred=best_letter_p2_model.predict(letter_p2_X_test))

    f = open("Results/best_letter_p2_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_letter_p2_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_letter_p2_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
Running trial: 2
Running trial: 3
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:1505:
UserWarning: Setting penalty='none' will ignore the C and l1_ratio parameters
  "Setting penalty='none' will ignore the C and l1_ratio "
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:1505:
UserWarning: Setting penalty='none' will ignore the C and l1_ratio parameters
  "Setting penalty='none' will ignore the C and l1_ratio "
```

```
CPU times: user 6.31 s, sys: 7.83 s, total: 14.1 s
Wall time: 8.99 s
```

Random Forests Modeling

December 16, 2020

0.1 Random Forests Machine

For Random Forest, this is the outline of what I will be doing

[1]:
```
# MAKE SURE TO RUN THIS CELL BEFORE MAKING EDITS

# import libaries and set global variables
import os.path
import pandas as pd
import numpy as np
import joblib
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
```

[2]:
```
# read in the datasets for logistic regression
adult = pd.read_csv('../Data/adult_clean.csv')
cover = pd.read_csv('../Data/cover_clean.csv')
letter_p1 = pd.read_csv('../Data/letter_p1.csv')
letter_p2 = pd.read_csv('../Data/letter_p2.csv')
```

[3]:
```
# separate the datasets into the data (X) and labels (Y)
adult_X = adult.iloc[:,:-1]
adult_Y = adult.iloc[:,-1]

cover_X = cover.iloc[:,:-1]
cover_Y = cover.iloc[:,-1]

letter_p1_X = letter_p1.iloc[:,:-1]
letter_p1_Y = letter_p1.iloc[:,-1]

letter_p2_X = letter_p2.iloc[:,:-1]
letter_p2_Y = letter_p2.iloc[:,-1]
```

[4]:
```
# get regularization parameter values
num_trees = 1024

# get the radial width
feature_set = [1, 2, 4, 6, 8, 12, 16, 20]

# define the cross-validation
cv = StratifiedKFold(n_splits=5)

# build the pipeline
pipe = Pipeline([('classifier', RandomForestClassifier())])


# setting up the parameters
param = {'classifier__n_estimators': [num_trees],
         'classifier__max_features': feature_set}

# # create grid search
clf = GridSearchCV(estimator=pipe, param_grid=param, cv=cv, n_jobs=-1,␣
↪verbose=0)
```

0.1.1 Run RF on Adult dataset for 3 trials

[5]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_adult_results.txt"):
    os.remove("Results/best_adult_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for testing
    adult_X_train, adult_X_test, adult_Y_train, adult_Y_test =␣
↪train_test_split(adult_X,adult_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on adult dataset and save the model
    adult_model = clf.fit(adult_X_train, adult_Y_train)
    joblib.dump(adult_model, 'Models/adult_model_'+str(trial)+'.pkl',␣
↪compress=1)
    print("\tFit on CV folds")

    # fit best parameters to training set and save the model
    best_adult_model = adult_model.fit(adult_X_train, adult_Y_train)
```

```
    joblib.dump(best_adult_model, 'Models/best_adult_model_'+str(trial)+'.pkl',␣
↪compress=1)
    print("\tFit on training set")

    # get the accuracy score for the testing and training data
    train_acc = accuracy_score(y_true=adult_Y_train, y_pred=best_adult_model.
↪predict(adult_X_train))
    test_acc = accuracy_score(y_true=adult_Y_test, y_pred=best_adult_model.
↪predict(adult_X_test))
    print("\tPredicted accuracy scores")

    # write the accuracy score into a file
    f = open("Results/best_adult_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_adult_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_adult_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 2
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 3
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
CPU times: user 1min 14s, sys: 1.24 s, total: 1min 15s
Wall time: 4min 16s
```

0.1.2 Run RF on Cover dataset for 3 trials

[6]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_cover_results.txt"):
    os.remove("Results/best_cover_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)    # print the trial number
```

```
    # split the data into 5000 points for training and save the rest for testing
    cover_X_train, cover_X_test, cover_Y_train, cover_Y_test =␣
↪train_test_split(cover_X,cover_Y,
                                                                    ␣
↪train_size=5000, random_state=trial)

    # fit the grid search on cover dataset and save the model
    cover_model = clf.fit(cover_X_train, cover_Y_train)
    joblib.dump(cover_model, 'Models/cover_model_'+str(trial)+'.pkl',␣
↪compress=1)
    print("\tFit on CV folds")

    # fit best parameters to training set and save the model
    best_cover_model = cover_model.fit(cover_X_train, cover_Y_train)
    joblib.dump(best_cover_model, 'Models/best_cover_model_'+str(trial)+'.pkl',␣
↪compress=1)
    print("\tFit on training set")

    # get the accuracy score for the testing and training data
    train_acc = accuracy_score(y_true=cover_Y_train, y_pred=best_cover_model.
↪predict(cover_X_train))
    test_acc = accuracy_score(y_true=cover_Y_test, y_pred=best_cover_model.
↪predict(cover_X_test))
    print("\tPredicted accuracy scores")

    # write the accuracy score into a file
    f = open("Results/best_cover_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_cover_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_cover_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 2
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 3
        Fit on CV folds
        Fit on training set
```

```
        Predicted accuracy scores
CPU times: user 6min 25s, sys: 1.78 s, total: 6min 27s
Wall time: 11min 34s
```

0.1.3 Run RF on Letter_p1 dataset for 3 trials

[7]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_letter_p1_results.txt"):
    os.remove("Results/best_letter_p1_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for␣
 ↪testing
    letter_p1_X_train, letter_p1_X_test, letter_p1_Y_train, letter_p1_Y_test =␣
 ↪train_test_split(letter_p1_X,letter_p1_Y,
                                                                          ␣
 ↪train_size=5000, random_state=trial)

    # fit the grid search on letter_p1 dataset and save the model
    letter_p1_model = clf.fit(letter_p1_X_train, letter_p1_Y_train)
    joblib.dump(letter_p1_model, 'Models/letter_p1_model_'+str(trial)+'.pkl',␣
 ↪compress=1)
    print("\tFit on CV folds")

    # fit best parameters to training set and save the model
    best_letter_p1_model = letter_p1_model.fit(letter_p1_X_train,␣
 ↪letter_p1_Y_train)
    joblib.dump(best_letter_p1_model, 'Models/
 ↪best_letter_p1_model_'+str(trial)+'.pkl', compress=1)
    print("\tFit on training set")

    # get the accuracy score for the testing and training data
    train_acc = accuracy_score(y_true=letter_p1_Y_train,␣
 ↪y_pred=best_letter_p1_model.predict(letter_p1_X_train))
    test_acc = accuracy_score(y_true=letter_p1_Y_test,␣
 ↪y_pred=best_letter_p1_model.predict(letter_p1_X_test))
    print("\tPredicted accuracy scores")

    f = open("Results/best_letter_p1_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
 ↪best_letter_p1_model.best_score_))
```

```
    f.write('\tBest Parameters: %s\n' % best_letter_p1_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 2
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 3
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
CPU times: user 41.9 s, sys: 245 ms, total: 42.2 s
Wall time: 3min 2s
```

0.1.4 Run RF on Letter_p2 datasets for 3 trials

[8]:
```
%%time

# remove results file if it exists
if os.path.isfile("Results/best_letter_p2_results.txt"):
    os.remove("Results/best_letter_p2_results.txt")

# run for 3 trials
for trial in [1,2,3]:
    print("Running trial:",trial)

    # split the data into 5000 points for training and save the rest for␣
 ↪testing
    letter_p2_X_train, letter_p2_X_test, letter_p2_Y_train, letter_p2_Y_test =␣
 ↪train_test_split(letter_p2_X,letter_p2_Y,
                                                                          ␣
 ↪train_size=5000, random_state=trial)

    # fit the grid search on letter_p2 dataset and save the model
    letter_p2_model = clf.fit(letter_p2_X_train, letter_p2_Y_train)
    joblib.dump(letter_p2_model, 'Models/letter_p2_model_'+str(trial)+'.pkl',␣
 ↪compress=1)
    print("\tFit on CV folds")

    # fit best parameters to training set and save the model
```

```
    best_letter_p2_model = letter_p2_model.fit(letter_p2_X_train,␣
↪letter_p2_Y_train)
    joblib.dump(best_letter_p2_model, 'Models/
↪best_letter_p2_model_'+str(trial)+'.pkl', compress=1)
    print("\tFit on training set")

    # get the accuracy score for the testing and training data
    train_acc = accuracy_score(y_true=letter_p2_Y_train,␣
↪y_pred=best_letter_p2_model.predict(letter_p2_X_train))
    test_acc = accuracy_score(y_true=letter_p2_Y_test,␣
↪y_pred=best_letter_p2_model.predict(letter_p2_X_test))
    print("\tPredicted accuracy scores")

    f = open("Results/best_letter_p2_results.txt", "a")
    f.write('Trial '+str(trial)+":\n")
    f.write('\tAccuracy %.2f%% (average over CV test folds)\n' % (100 *␣
↪best_letter_p2_model.best_score_))
    f.write('\tBest Parameters: %s\n' % best_letter_p2_model.best_params_)
    f.write('\tTraining Accuracy: %.2f%%\n' % (100 * train_acc))
    f.write('\tTest Accuracy: %.2f%%\n\n' % (100 * test_acc))
    f.close()
```

```
Running trial: 1
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 2
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
Running trial: 3
        Fit on CV folds
        Fit on training set
        Predicted accuracy scores
CPU times: user 1min, sys: 555 ms, total: 1min
Wall time: 4min 58s
```

Results and Analysis

December 16, 2020

0.1 Accuracy Scores and Analysis

This notebook is used to gather the training and testing accuracy scores. The mean scores by dataset and overall scores are calculated in this notebook.

Finally, t-tests are performed for each trial of each dataset by algorithm comparison, as well as an overall t-test for algorithm comparison

[1]:
```
# import necessary packages
import pandas as pd
import numpy as np
from scipy import stats

# store results up to 4 decimal places
pd.options.display.float_format = "{:,.4f}".format
```

Raw training scores for each algorithm by dataset and trial number

[2]:
```
# training accuracies found in the "Results" directory categorized as "Training␣
↪Accuracy"
t = {'Algorithm':␣
↪['Adult1','Adult2','Adult3','Cover1','Cover2','Cover3','Letter_p11','Letter_p12','Letter_p1
    'KNN':        [100     , 100     , 100     ,  100    , 100     , 100     , 100  ␣
↪     , 100         , 100         , 100         , 100         , 100         ],
    'LOGIT':      [85.00    , 85.04   , 84.90   ,  76.44 , 76.50   , 75.38   , 96.56␣
↪     , 96.20       , 96.08       , 73.82       , 72.20       , 73.26       ],
    'RF':         [100     , 100     , 100     ,  100    , 100     , 100     , 100  ␣
↪     , 100         , 100         , 100         , 100         , 100         ]
   }

train_scores = pd.DataFrame(t).set_index('Algorithm').T.div(100)
train_scores
```

[2]:
```
Algorithm  Adult1  Adult2  Adult3  Cover1  Cover2  Cover3  Letter_p11  \
KNN        1.0000  1.0000  1.0000  1.0000  1.0000  1.0000      1.0000
LOGIT      0.8500  0.8504  0.8490  0.7644  0.7650  0.7538      0.9656
RF         1.0000  1.0000  1.0000  1.0000  1.0000  1.0000      1.0000

Algorithm  Letter_p12  Letter_p13  Letter_p21  Letter_p22  Letter_p23
```

```
KNN            1.0000      1.0000      1.0000      1.0000      1.0000
LOGIT          0.9620      0.9608      0.7382      0.7220      0.7326
RF             1.0000      1.0000      1.0000      1.0000      1.0000
```

Mean training scores for each algorithm by dataset

[3]:
```
# get the mean accuracy scores from the raw training data
ts = {'Algorithm': ['Adult', 'Cover', 'Letter_p1', 'Letter_p2'],
      'KNN':       [train_scores.iloc[0,:3].mean(), train_scores.iloc[0,3:6].
 ↪mean(), train_scores.iloc[0,6:9].mean(), train_scores.iloc[0,9:].mean()],
      'LOGIT':     [train_scores.iloc[1,:3].mean(), train_scores.iloc[1,3:6].
 ↪mean(), train_scores.iloc[1,6:9].mean(), train_scores.iloc[1,9:].mean()],
      'RF':        [train_scores.iloc[2,:3].mean(), train_scores.iloc[2,3:6].
 ↪mean(), train_scores.iloc[2,6:9].mean(), train_scores.iloc[2,9:].mean()]
     }
mean_train = pd.DataFrame(ts).set_index('Algorithm').T
mean_train
```

[3]:
```
Algorithm  Adult  Cover  Letter_p1  Letter_p2
KNN       1.0000 1.0000     1.0000     1.0000
LOGIT     0.8498 0.7611     0.9628     0.7309
RF        1.0000 1.0000     1.0000     1.0000
```

Raw testing scores for each algorithm by dataset and trial number

[4]:
```
# testing accuracies found in the "Results" directory categorized as "Testing␣
 ↪Accuracy"
s = {'Algorithm':␣
 ↪['Adult1','Adult2','Adult3','Cover1','Cover2','Cover3','Letter_p11','Letter_p12','Letter_p1
     'KNN':       [82.99   , 82.24   , 82.51   ,  78.35 , 77.66   , 78.19   , 99.11␣
 ↪      , 99.13       , 99.09       , 95.53       , 95.67       , 95.83],
     'LOGIT':     [84.61   , 84.54   , 84.45   ,  75.48 , 75.40   , 76.13   , 96.13␣
 ↪      , 96.25       , 96.29       , 72.65       , 72.59       , 72.70],
     'RF':        [84.78   , 84.41   , 84.61   ,  82.13 , 81.68   , 82.64   , 98.53␣
 ↪      , 98.93       , 98.77       , 94.36       , 94.99       , 94.31]
    }

raw_scores = pd.DataFrame(s).set_index('Algorithm').T.div(100)
raw_scores
```

[4]:
```
Algorithm  Adult1  Adult2  Adult3  Cover1  Cover2  Cover3  Letter_p11  \
KNN        0.8299  0.8224  0.8251  0.7835  0.7766  0.7819      0.9911
LOGIT      0.8461  0.8454  0.8445  0.7548  0.7540  0.7613      0.9613
RF         0.8478  0.8441  0.8461  0.8213  0.8168  0.8264      0.9853

Algorithm  Letter_p12  Letter_p13  Letter_p21  Letter_p22  Letter_p23
KNN            0.9913      0.9909      0.9553      0.9567      0.9583
```

```
LOGIT          0.9625      0.9629      0.7265      0.7259      0.7270
RF             0.9893      0.9877      0.9436      0.9499      0.9431
```

Mean testing scores for each algorithm by dataset

[5]:
```
# get the mean accuracy scores from the raw testing data
ms = {'Algorithm': ['Adult', 'Cover', 'Letter_p1', 'Letter_p2'],
      'KNN':       [raw_scores.iloc[0,:3].mean(), raw_scores.iloc[0,3:6].
 ↪mean(), raw_scores.iloc[0,6:9].mean(), raw_scores.iloc[0,9:].mean()],
      'LOGIT':     [raw_scores.iloc[1,:3].mean(), raw_scores.iloc[1,3:6].
 ↪mean(), raw_scores.iloc[1,6:9].mean(), raw_scores.iloc[1,9:].mean()],
      'RF':        [raw_scores.iloc[2,:3].mean(), raw_scores.iloc[2,3:6].
 ↪mean(), raw_scores.iloc[2,6:9].mean(), raw_scores.iloc[2,9:].mean()]
     }
mean_scores = pd.DataFrame(ms).set_index('Algorithm').T
mean_scores
```

[5]:
```
Algorithm  Adult  Cover  Letter_p1  Letter_p2
KNN       0.8258 0.7807     0.9911     0.9568
LOGIT     0.8453 0.7567     0.9622     0.7265
RF        0.8460 0.8215     0.9874     0.9455
```

Mean testing scores for each algorithm overall

[6]:
```
# get the mean accuracy scores for each algorithm
acs = {'Algorithm': ['Accuracy'],
      'KNN':       [raw_scores.iloc[0,:].mean()],
      'LOGIT':     [raw_scores.iloc[1,:].mean()],
      'RF':        [raw_scores.iloc[2,:].mean()]
     }
mean_scores = pd.DataFrame(acs).set_index('Algorithm').T
mean_scores
```

[6]:
```
Algorithm  Accuracy
KNN          0.8886
LOGIT        0.8227
RF           0.9001
```

0.1.1 Calculate t-test

[7]:
```
def ttest(a,b):
    '''
    A function to evaluate the t-test on two algorithms for each trial per␣
 ↪dataset
    '''
    li = []
    for i in range(raw_scores.shape[0]+1):
```

```
        li.append(stats.ttest_rel(raw_scores.iloc[a,i*3:i*3+3], raw_scores.
↪iloc[b,i*3:i*3+3]))
    return li

ab = pd.DataFrame(ttest(0,1), columns=['t-test','p-value'])
bc = pd.DataFrame(ttest(1,2), columns=['t-test','p-value'])
ac = pd.DataFrame(ttest(0,2), columns=['t-test','p-value'])
```

0.1.2 T-test for algorithm by datasets

[8]:
```
# get the t-statistic for each algorithm comparison by dataset
dat = {'Algorithm':    ['Adult'     , 'Cover'     , 'Letter_p1' , 'Letter_p2'],
       'KNN / LOGIT':  [ab.iloc[0,0], ab.iloc[1,0], ab.iloc[2,0], ab.iloc[3,0]],
       'LOGIT / RF':   [bc.iloc[0,0], bc.iloc[1,0], bc.iloc[2,0], bc.iloc[3,0]],
       'RF / KNN':     [ac.iloc[0,0], ac.iloc[1,0], ac.iloc[2,0], ac.iloc[3,0]],
      }

data_alg_t = pd.DataFrame(dat).set_index('Algorithm').T
data_alg_t
```

[8]:

Algorithm	Adult	Cover	Letter_p1	Letter_p2
KNN / LOGIT	-9.9451	9.8382	55.4400	301.5335
LOGIT / RF	-0.6777	-60.0790	-30.2642	-88.2088
RF / KNN	-17.3002	-20.8347	3.2694	4.6112

[9]:
```
# get the p-value for each algorithm comparison by dataset
dap = {'Algorithm': ['Adult'     , 'Cover'     , 'Letter_p1' , 'Letter_p2'],
       'KNN':       [ab.iloc[0,1], ab.iloc[1,1], ab.iloc[2,1], ab.iloc[3,1]],
       'LOGIT':     [bc.iloc[0,1], bc.iloc[1,1], bc.iloc[2,1], bc.iloc[3,1]],
       'RF':        [ac.iloc[0,1], ac.iloc[1,1], ac.iloc[2,1], ac.iloc[3,1]],
      }

data_alg_p = pd.DataFrame(dap).set_index('Algorithm').T
data_alg_p
```

[9]:

Algorithm	Adult	Cover	Letter_p1	Letter_p2
KNN	0.0100	0.0102	0.0003	0.0000
LOGIT	0.5679	0.0003	0.0011	0.0001
RF	0.0033	0.0023	0.0822	0.0440

0.1.3 T-test by algorithm

[10]:
```
# get the t-statistic for each algorithm comparison
at = {'Algorithm': ['T-statistic'],
      'KNN':       [stats.ttest_rel(raw_scores.iloc[0,:],raw_scores.iloc[1,:
↪])[0]],
```

```
      'LOGIT':     [stats.ttest_rel(raw_scores.iloc[1,:],raw_scores.iloc[2,:
 ↪])[0]],
      'RF':        [stats.ttest_rel(raw_scores.iloc[0,:],raw_scores.iloc[2,:
 ↪])[0]]
     }

alg_t = pd.DataFrame(at).set_index('Algorithm').T
alg_t
```

[10]:
```
Algorithm  T-statistic
KNN             2.2580
LOGIT          -3.0235
RF             -1.8513
```

[11]:
```
# get the p-value for each algorithm comparison
ap = {'Algorithm': ['P-value'],
      'KNN':       [stats.ttest_rel(raw_scores.iloc[0,:],raw_scores.iloc[1,:
 ↪])[1]],
      'LOGIT':     [stats.ttest_rel(raw_scores.iloc[1,:],raw_scores.iloc[2,:
 ↪])[1]],
      'RF':        [stats.ttest_rel(raw_scores.iloc[0,:],raw_scores.iloc[2,:
 ↪])[1]]
     }

alg_p = pd.DataFrame(ap).set_index('Algorithm').T
alg_p
```

[11]:
```
Algorithm  P-value
KNN         0.0452
LOGIT       0.0116
RF          0.0911
```