Caruana Replication: Cogs 118A Final Project

Kylie Morgan **kmmorgan@ucsd.edu**

Cognitive Science Major

University of California, San Diego

La Jolla, CA, 92092, USA

Abstract

This paper is based on an earlier project by Curana and Niculescu-Mizil. They iterate through eight algorithms and datasets to compare the relative abilities of each algorithm. This paper guides users to choose algorithms and consider what the different classifiers output: probabilities, class labels, etc. These are compared across different binary classification datasets and different metrics. For the individual version of this project, only three classification algorithms, four datasets, and three metric measurements are required. Caruana gives the groundwork of how to format the information coming from the algorithms to render them comparable. Statistical analysis is used to further represent the merits of this approach.

1. Introduction

Caruana was one of the first large scale papers comparing learning algorithms. This class is specialized to learning algorithms in supervised learning; supervised learning means the algorithm learns the data with class labels attached. Classification draws a boundary in order to best divide the classes across the boundary. I evaluate the performance of SVMs, Logistic Regression, and K nearest neighbors. These algorithms are compared on the performance metrics F-score, accuracy, and ROC area. SVM maximizes the margin between the decision boundary line and the positive/negative class lines (Fleischer). In order to account for linearly inseparable data, SVM varies over regularization parameter C. Apart from linear SVM, I also analyze polynomial SVM, degrees 2 and 3, and rbf SVM, which uses lagrangian math to determine optimal support vectors. These decision boundaries aren't linear. For Logistic regression, the algorithm is outputting a probability of a datapoint belonging to a class. This is helpful to know the confidence in your prediction. Logistic regression utilizes the sigmoid logistic function to have a loss function that is differentiable everywhere (Fleischer). KNN is very different from these two. With fast training time, but very slow testing time, this algorithm compares the distance of a testing point to the distances of all training points. For KNN, I iterate over both Euclidean distance and Manhattan distance. KNN labels a point based on the class labels of the K nearest training neighbors.

2. Method

2.1. Learning Algorithms

SVMs: I use the following kernels: linear, polynomial degree 2 & 3, rbf with gamma {0.001,0.005,0.01,0.05,0.1,0.5,1,2}. I changed the regularization parameter, C, by factors of ten from 10−7 to 103 for each kernel.

Logistic Regression (LOGREG): I trained with both unregularized and regularized models. For regularization, I used both L1 and L2. The solver also varied from 'saga' to 'lbfgs'; keeping the max iterations at 5000. The regularization parameter varied by factors of 10 from 10^−8 to 10^4.

K-Nearest Neighbor(KNN): I used 5 values of K; 5, 10, 20, 200, 1000; this was different from the Caruana paper and was done to help lower computation time. I began to run out of time towards the end of this project. I use KNN with Euclidean distance and Manhattan distance. The weights were also varied: uniform and distance.

2.2. Datasets

To begin, I download the three datasets from the UCI repository; the datasets are titled LETTER, ADULT, and COV_TYPE. To create a binary classification problem, for the LETTER dataset, I divided the twenty six classes into two. For the first problem, the letter 'O' was labeled as the positive class and the rest of the letters were negative. In the second problem, letters 'A'- 'M' are labeled as positive and 'N'- 'Z' as negative. Since the first problem is not evenly distributed, I made sure to use stratified kfold. This created two of the binary datasets that I would test on. For the second dataset, COV-TYPE, I read through the data info writeup and labeled the most popular class, group 2, as the positive class. I also renamed the columns to better understand the data features being used. For the Adult dataset, I used pandas 'get_dummies' function to one hot encode categorical variables. I also changed the salary income to get a binary classification problem. Table 1 shows a breakdown of the datasets.

Problem	# ATTR	Train size	Test Size	% Positive Class
Letter 1	16	5000	14000	3%
Letter 2	16	5000	14000	53%
Adult	14/104	5000	35222	25%
Cov_type	54	5000	25000	36%

Figure 1: Problem set breakdown: Number of features/attributes, train and test size, and the percent of data that belongs to the positive class.

3. Experiment

From there, I began by creating three notebooks, one for each type of classifier, so that the notebooks could run in parallel. Each classifier and each dataset had 5 trials with 5 folds of cross validation. For each trial, I selected five thousand samples to be the training and validation set. I then created a pipeline to standardize the data. For each algorithm, I created a search space to iterate through the hyperparameters, but avoid incompatible hyperparameters. The train and validation data was used for five cross validations and produced three sets of hyperparameters for each metric: F1 Micro score, accuracy, and roc auc curve. These were saved into a list. I also saved the mean scores into another list. After the five trials, each list had fifteen items. Table 2, just like Caruana, shows the normalized score for each algorithm on my three selected metrics. For each problem and metric we find the best parameter settings for each algorithm using the 1k validation sets set aside by cross- validation, then report that model's normalized score on the final test set.

Model	METRIC	LETTER 1	LETTER 2	ADULT	COV-TYPE	MEAN
SVM	ROC	**0.99746**	**0.991119***	**1***	**0.8591**	**0.96192**
KNN	ROC	0.9931	0.9883	0.982	0.8561*	0.954875*
SVM	ACC	0.99264	0.95528	**1***	0.79152	0.93486
SVM	F1 Micro	0.99264	0.95528	**1***	0.79152	0.93486
KNN	ACC	0.98956	0.94416	0.97676	0.77808	0.92214
KNN	F1 Micro	0.98956	0.94416	0.97676	0.77808	0.92214
LOG REG	ROC	0.85567966	0.8137	**1***	0.815958	0.87133
LOG REG	F1 Micro	0.9632	0.7261	**1***	0.75032	0.859905
LOG REG	ACC	0.9632	0.726	**1***	0.75032	0.85988

Table 2:Normalized scores for each learning algorithm by Problem (divide over three metrics)

In the table, higher algorithms indicate better performance. The last column, MEAN, is the mean normalized score over the three metrics, four problems, and five trials. The models in the table are sorted by the mean normalized score in this column. In the table, the algorithm with the best performance on each metric is boldfaced. Interestingly, SVM and KNN had the same average score for accuracy; this was completely random as their scores for accuracy within a problem were not the same. Overall, SVM had the best mean score, KNN the second best and Logistic Regression had the third. The data points with an * were found to be statistically significant at the p = .05 level. I had to do an unpaired t test because I did not assign the same seed for each trial across algorithms. The ADULT problem set t-test was corrupted by the excessive number of ones received. I compared each mean to the highest mean for ever algorithms, problem set and metric with p values for each output; the p values are in appendix b. With very low standard deviations, not many of them were found to be statistically significant.

MODEL	ACC	ROC	F1 Micro	MEAN
SVM	**.93486***	**.96192**	.93486	**.943873***
KNN	**.93486***	.92214	**.954875**	.9372917*
LOG REG	.85988	.87133	.859905	.863705

Table 3:Normalized scores of each learning algorithm by metric (averaged over four problem sets)

Table 3 shows the normalized score for each algorithm on each of the 3 test metrics. Each entry is an average over the four datasets and five trials when selection is done using 1k validation sets.

4. Discussion

For both SVM and Logistic Regression, I received a score of 1 for the adult dataset metrics. I ran these algorithms after KNN, so I was not aware of an issue with the ADULT dataset cleaning I did. These performance metrics are too high for chance. Overall the algorithms ran better on LETTER and ADULT datasets. The specific results depend on the metric and problem, but overall SVM performed the best. The bootstrap analysis complements the t-tests in Tables 2 and 3. The results suggest that if we had selected other problems/metrics, there is a large chance that SVM would still have ranked 1st. Overall it is important to understand the No Free Lunch theorem; there is no one general purpose algorithm. It is important to understand the problem and what you want to get out of it.

5. References

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

J. Fleischer. Support Vector Machines. COGS 118A (Winter 2021).

J. Fleischer. Logistic Regression. COGS 118A (Winter 2021).

0.9634	0.8408456475	0.9634	0.963	0.8585445258	0.963	0.9628	0.8626520804	0.9628	0.9624	0.8638893803	0.9624	0.9644	0.8524666883	0.9644
0.7197409409	0.8079984335	0.7197409409	0.7273455455	0.8137284636	0.7273455455	0.7287461461	0.8193739547	0.7287461461	0.7283469469	0.8107588244	0.7283469469	0.7263427427	0.8166417612	0.7263427427
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.754	0.8289365333	0.754	0.7546	0.8223653138	0.7546	0.7472	0.81357207	0.7472	0.7434	0.8167249647	0.7434	0.7524	0.8162914607	0.7524
0.9918	0.9973695291	0.9918	0.9926	0.9965276692	0.9926	0.9938	0.9979756048	0.9938	0.9936	0.9982742931	0.9936	0.9914	0.9971614846	0.9914
0.9574	0.9915589324	0.9574	0.9564	0.9914227406	0.9564	0.9542	0.9906445444	0.9542	0.954	0.9911424653	0.954	0.9544	0.9908261667	0.9544
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

0.7896	0.8547878387	0.7896	0.7828	0.8561640995	0.7828	0.798	0.8658867286	0.798	0.7976	0.8629003681	0.7976	0.7896	0.8557175282	0.7896
0.9878	0.9953523361	0.9878	0.9898	0.9874081443	0.9898	0.9904	0.9940702748	0.9904	0.9896	0.9956245103	0.9896	0.9902	0.9930093777	0.9902
0.9442	0.9901782461	0.9442	0.942	0.9868975013	0.942	0.9458	0.9874528258	0.9458	0.9438	0.988817977	0.9438	0.945	0.9881637738	0.945
0.9438	0.9817093088	0.9438	0.9458	0.9818624835	0.9458	0.9492	0.981608295	0.9492	0.9496	0.982932191	0.9496	0.9454	0.9817632257	0.9454
0.7714	0.8486835497	0.7714	0.7796	0.856661134	0.7796	0.7756	0.8527088741	0.7756	0.7794	0.8586781336	0.7794	0.7844	0.8639497428	0.7844

Appendix A

P VALUES	METRIC	LETTER 1	LETTER 2	ADULT	COV-TYPE	MEAN
P VALUES	SVM ROC	**1***	**1***	**1***	**1**	**1**
P VALUES	KNN ROC	< 0.005	.009	< 0.005	.13	< 0.005
P VALUES	SVM ACC	< 0.005	< 0.005	**1***	< 0.005	< 0.005
P VALUES	SVM F1 Micro	< 0.005	< 0.005	**1***	< 0.005	< 0.005
P VALUES	KNN ACC	0.0095	< 0.005	.03	< 0.005	< 0.005
P VALUES	KNN F1 Micro	0.0095	< 0.005	.03	< 0.005	< 0.005
P VALUES	LOG REG ROC	< 0.005	< 0.005	**1***	< 0.005	< 0.005
P VALUES	LOG REG F1 Micro	< 0.005	< 0.005	**1***	< 0.005	< 0.005
P VALUES	LOG REG ACC	< 0.005	< 0.005	**1***	< 0.005	< 0.005

Appendix B: P Values for table 2

P VALUES	MODEL	ACC	ROC	F1 Micro	MEAN
P VALUES	SVM	**1***	< 0.005	< 0.005	1*
P VALUES	KNN	**1***	< 0.005	**1**	.06*
P VALUES	LOG REG	< 0.005	< 0.005	< 0.005	< 0.005

Appendix C: P values for table 3

Final Project Logistic Regression

March 16, 2021

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set_style('white')  # plot formatting
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import validation_curve
from sklearn.metrics import r2_score
from sklearn.model_selection import learning_curve


from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

0.1 Logistic Regression

Letter 1 dataset

[2]:
```python
#import letter data set
#letter1 for the problem with 'O' as the postive class rest as negative
#letter2, for the problem with A-M as positive and N-Z as negative
letter = pd.read_csv('letter-recognition.data')

letter_ = letter.replace('O', +1)
```

```
letter1 = letter_.replace(to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',␣
↪'I', 'J', 'K', 'L', 'M', 'N',
                                           'P', 'Q', 'R', 'S', 'T', 'U', 'V',␣
↪'W', 'X', 'Y', 'Z'], value = -1)
letter_Y = letter.replace (to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], value = +1)
letter2 = letter_Y.replace(to_replace = ['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], value = -1)

X_1 = letter1
Y_1 = X_1[['T']]
X_1 = X_1.iloc[:, 1:]

X_2 = letter2
Y_2 = X_2[['T']]
X_2 = X_2.iloc[:, 1:]
```

[9]:
```
optimal_hyperparameters_1 = []
mean_scores_1 = []

for i in range(5):
    #split the data so that training size is 5000
    X_train, X_test, y_train, y_test = train_test_split(X_1, Y_1, train_size =␣
↪5000)



            # Create a pipeline - RF is a stand in, we will populate the␣
↪classifier part below
    pipe_log = Pipeline([('std', StandardScaler()),
                 ('classifier', LogisticRegression())])

            # Create search space of candidate learning algorithms and their␣
↪hyperparameters
            # note lbfgs can't do l1, and if you pass penalty='none' it expects␣
↪no C value
    search_space_log = [{'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['saga'],
                 'classifier__penalty': ['l1', 'l2'],
                 'classifier__C': [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2,␣
↪1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]},

                {'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['lbfgs'],
                 'classifier__penalty': ['l2'],
                 'classifier__C': [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2,␣
↪1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]},
```

```
                {'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['lbfgs','saga'],
                 'classifier__penalty': ['none']}
                ]

            # Create grid search
    clf_log = GridSearchCV(pipe_log, search_space_log,␣
↪cv=StratifiedKFold(n_splits=5),
                    scoring=['accuracy', 'roc_auc', 'f1_micro'], refit=False,
                    verbose=0)

            # Fit grid search
    best_model_log = clf_log.fit(X_train, y_train)


            # output best hyperparameter set indexed at best metric scores
    h1 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_1.append(h1)
    optimal_hyperparameters_1.append(h2)
    optimal_hyperparameters_1.append(h3)
    mean_scores_1.append(best_model_log.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_log.cv_results_['rank_test_accuracy'])])
    mean_scores_1.append(best_model_log.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_log.cv_results_['rank_test_roc_auc'])])
    mean_scores_1.append(best_model_log.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_log.cv_results_['rank_test_f1_micro'])])


print(optimal_hyperparameters_1)
print(mean_scores_1)
```

```
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
```

```
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

```
[{'classifier': LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
```

```
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1e-08,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}]
[0.9634, 0.8408456474856212, 0.9634, 0.9629999999999999, 0.8585445258342455,
0.9629999999999999, 0.9628, 0.8626520803599013, 0.9628, 0.9623999999999999,
0.8638893803183031, 0.9623999999999999, 0.9644, 0.8524666882684103, 0.9644]
```

```
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

0.2 Logistic Regression

Letter 2 dataset

[4]:
```
optimal_hyperparameters_2 = []
mean_scores_2 = []
for i in range(5):
    X_train, X_test, y_train, y_test = train_test_split(X_2, Y_2, train_size = .
↪25)

            # Fit grid search
    best_model_log = clf_log.fit(X_train, y_train)


            # output best hyperparameter set indexed at best metric scores
    h1 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_2.append(h1)
    optimal_hyperparameters_2.append(h2)
    optimal_hyperparameters_2.append(h3)
    mean_scores_2.append(best_model_log.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_log.cv_results_['rank_test_accuracy'])])
    mean_scores_2.append(best_model_log.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_log.cv_results_['rank_test_roc_auc'])])
    mean_scores_2.append(best_model_log.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_log.cv_results_['rank_test_f1_micro'])])
```

```python
print(optimal_hyperparameters_2)
print(mean_scores_2)
```

```
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
```

```
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

```
[{'classifier': LogisticRegression(max_iter=5000), 'classifier__C': 0.1,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__penalty': 'none',
'classifier__solver': 'saga'}, {'classifier': LogisticRegression(max_iter=5000),
'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver':
'saga'}, {'classifier': LogisticRegression(max_iter=5000), 'classifier__C': 0.1,
```

```
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 100.0,
'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10.0, 'classifier__penalty':
'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}]
[0.7197409409409409, 0.8079984334851341, 0.7197409409409409, 0.7273455455455455,
0.8137284636101553, 0.7273455455455455, 0.7287461461461462, 0.819373954693854,
0.7287461461461462, 0.7283469469469469, 0.8107588243949992, 0.728346946946947,
0.7263427427427428, 0.8166417611893586, 0.7263427427427428]
```

0.3 Logistic Regression

Adult dataset

[5]:
```python
adult_main = pd.read_csv('adult.data')
adult_test = pd.read_csv ('adult.test')
adult = pd.concat([adult_main,adult_test], ignore_index = True)
#make class column binary
adult_ = adult.replace(' >50K', -1)
adult = adult_.replace(' <=50K', +1)

#move class column to be in 0th index
adult = adult[[ ' <=50K','39', ' State-gov', ' 77516', ' Bachelors', ' 13', '␣
↪Never-married',
       ' Adm-clerical', ' Not-in-family', ' White', ' Male', ' 2174', ' 0',
       ' 40', ' United-States', '|1x3 Cross validator']]

#rename columns according to data write-up
adult = adult.rename(columns={" <=50K": "Y", '39': 'age', ' State-gov':␣
↪'workclass', ' 77516': 'fnlwgt',
```

```
                              ' Bachelors': 'education', ' 13':␣
↪'education-num', ' Never-married': 'marital-status',
                              ' Adm-clerical':'occupation', ' Not-in-family' :␣
↪'relationship', ' White' : 'race',
                              ' Male' : 'sex', ' 2174' : 'capital-gain', ' 0' :␣
↪'capital-loss',' 40': 'hours-per-week',
                              ' United-States' : 'native-country'})

#one hot encode via pd.get_dummies
work = pd.get_dummies(adult['workclass'])
work.rename(columns = {' ?': 'NA_Work'}, inplace = True)

education = pd.get_dummies(adult['education'])
marital = pd.get_dummies(adult['marital-status'])
occu = pd.get_dummies(adult['occupation'])
occu.rename(columns = {' ?': 'NA_Occu'}, inplace = True)

relationship = pd.get_dummies(adult['relationship'])
race = pd.get_dummies(adult['race'])
sex = pd.get_dummies(adult['sex'])
country = pd.get_dummies(adult['native-country'])
country.rename(columns = {' ?': 'NA_Country'}, inplace = True)\

adult_= pd.concat([work,education,marital,occu,relationship,race,sex,country],␣
↪axis = 1)

#combine the continuous and categorical (now one hot encoded variables)
adult_ = pd.concat([adult[['Y',␣
↪'age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']],adult_],␣
↪axis =1)

adult_ = adult_[adult_.NA_Work == 0]
adult_ = adult_[adult_.NA_Occu == 0]
adult_ = adult_[adult_.NA_Country == 0]

adult_.drop(['NA_Work','NA_Occu','NA_Country'],axis = 1, inplace=True)

adult_.reset_index(inplace = True)
adult_.dropna(inplace = True)

#split off class data into seperate grouping
X_3 = adult_
Y_3 = X_3[['Y']]
X_3 = X_3.iloc[:, 1:]
```

[6]:
```
optimal_hyperparameters_3 = []
mean_scores_3 = []

for i in range(5):
    X_train, X_test, y_train, y_test = train_test_split(X_3, Y_3, train_size =␣
 ↪5000)

            # Fit grid search
    best_model_log = clf_log.fit(X_train, y_train)


            # output best hyperparameter set indexed at best metric scores
    h1 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
 ↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
 ↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
 ↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_3.append(h1)
    optimal_hyperparameters_3.append(h2)
    optimal_hyperparameters_3.append(h3)
    mean_scores_3.append(best_model_log.cv_results_['mean_test_accuracy'][np.
 ↪argmin(best_model_log.cv_results_['rank_test_accuracy'])])
    mean_scores_3.append(best_model_log.cv_results_['mean_test_roc_auc'][np.
 ↪argmin(best_model_log.cv_results_['rank_test_roc_auc'])])
    mean_scores_3.append(best_model_log.cv_results_['mean_test_f1_micro'][np.
 ↪argmin(best_model_log.cv_results_['rank_test_f1_micro'])])

print(optimal_hyperparameters_3)
print(mean_scores_3)
```

```
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

```
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

```
[{'classifier': LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
```

```
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.001,
'classifier__penalty': 'l1', 'classifier__solver': 'saga'}]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

0.4 Logistic Regression

Cov-type dataset

[7]:
```
#read in data move the class column to the 0th column
#rename columns (last 44 columns = leave one out arrangement)
cov_type = pd.read_csv('covtype.data')
cov_type = cov_type[['5', '2596', '51', '3', '258', '0', '510', '221', '232',␣
↪'148', '6279', '1',
       '0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9', '0.10',
       '0.11', '0.12', '0.13', '0.14', '0.15', '0.16', '0.17', '0.18', '0.19',
       '0.20', '0.21', '0.22', '0.23', '0.24', '0.25', '0.26', '0.27', '0.28',
       '0.29', '0.30', '0.31', '1.1', '0.32', '0.33', '0.34', '0.35', '0.36',
       '0.37', '0.38', '0.39', '0.40', '0.41', '0.42']]
cov_type.rename(columns = {'5':'Y', '2596':'Elevation', '51': 'Aspect', '3':␣
↪'Slope', '258':
                           'Horizontal_Distance_To_Hydrology', '0':␣
↪'Vertical_Distance_To_Hydrology', '510':
                           'Horizontal_Distance_To_Roadways', '221' :␣
↪'Hillshade_9am', '232' : 'Hillshade_Noon',
                           '148': 'Hillshade_3pm ', '6279':␣
↪'Horizontal_Distance_To_Fire_Points'}, inplace = True)

#make the classes binary so the largest class (2) is +1, rest is -1
cov_type = cov_type.replace(to_replace = {'Y': {1: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {3: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {4: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {5: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {6: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {7: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {2: +1}})

X_4 = cov_type
Y_4 = X_4[['Y']]
X_4 = X_4.iloc[:, 1:]
```

[8]:
```
optimal_hyperparameters_4 = []
mean_scores_4 = []

for i in range(5):
```

```
    X_train, X_test, y_train, y_test = train_test_split(X_4, Y_4, train_size =␣
↪5000)

            # Fit grid search
    best_model_log = clf_log.fit(X_train, y_train)


            # output best hyperparameter set indexed at best metric scores
    h1 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_log.cv_results_['params'][ np.argmin(best_model_log.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_4.append(h1)
    optimal_hyperparameters_4.append(h2)
    optimal_hyperparameters_4.append(h3)
    mean_scores_4.append(best_model_log.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_log.cv_results_['rank_test_accuracy'])])
    mean_scores_4.append(best_model_log.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_log.cv_results_['rank_test_roc_auc'])])
    mean_scores_4.append(best_model_log.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_log.cv_results_['rank_test_f1_micro'])])

print(optimal_hyperparameters_4)
print(mean_scores_4)
```

```
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
```

```
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
/opt/conda/lib/python3.8/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

```
[{'classifier': LogisticRegression(max_iter=5000), 'classifier__C': 0.1,
'classifier__penalty': 'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.1, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.1, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10.0, 'classifier__penalty':
'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 0.01, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l2', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 1.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10000.0,
'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10.0, 'classifier__penalty':
'l1', 'classifier__solver': 'saga'}, {'classifier':
LogisticRegression(max_iter=5000), 'classifier__C': 10000.0,
```

```
'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs'}]
[0.7539999999999999, 0.8289365333429334, 0.7540000000000001, 0.7545999999999999,
0.8223653138363348, 0.7545999999999999, 0.7472, 0.8135720700493938, 0.7472,
0.7434000000000001, 0.816724964691866, 0.7434000000000001, 0.7524,
0.8162914606633708, 0.7524]
```

[10]:
```python
print(mean_scores_1)
print(mean_scores_2)
print(mean_scores_3)
print(mean_scores_4)
```

```
[0.9634, 0.8408456474856212, 0.9634, 0.9629999999999999, 0.8585445258342455,
0.9629999999999999, 0.9628, 0.8626520803599013, 0.9628, 0.9623999999999999,
0.8638893803183031, 0.9623999999999999, 0.9644, 0.8524666882684103, 0.9644]
[0.7197409409409409, 0.8079984334851341, 0.7197409409409409, 0.7273455455455455,
0.8137284636101553, 0.7273455455455455, 0.7287461461461462, 0.819373954693854,
0.7287461461461462, 0.7283469469469469, 0.8107588243949992, 0.728346946946947,
0.7263427427427428, 0.8166417611893586, 0.7263427427427428]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.7539999999999999, 0.8289365333429334, 0.7540000000000001, 0.7545999999999999,
0.8223653138363348, 0.7545999999999999, 0.7472, 0.8135720700493938, 0.7472,
0.7434000000000001, 0.816724964691866, 0.7434000000000001, 0.7524,
0.8162914606633708, 0.7524]
```

[ ]:
```python

```

Final Project SVM

March 16, 2021

0.1 SVM

letter data 1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set_style('white')  # plot formatting
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import validation_curve
from sklearn.metrics import r2_score
from sklearn.model_selection import learning_curve


from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

0.2 SVM

letter dataset 1

[2]:
```
#import letter data set
#letter1 for the problem with 'O' as the postive class rest as negative
#letter2, for the problem with A-M as positive and N-Z as negative
letter = pd.read_csv('letter-recognition.data')

letter_ = letter.replace('O', +1)
letter1 = letter_.replace(to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',␣
↪'I', 'J', 'K', 'L', 'M', 'N',
                                              'P', 'Q', 'R', 'S', 'T', 'U', 'V',␣
↪'W', 'X', 'Y', 'Z'], value = -1)
letter_Y = letter.replace (to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], value = +1)
letter2 = letter_Y.replace(to_replace = ['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], value = -1)

X_1 = letter1
Y_1 = X_1[['T']]
X_1 = X_1.iloc[:, 1:]

X_2 = letter2
Y_2 = X_2[['T']]
X_2 = X_2.iloc[:, 1:]
```

[3]:
```
optimal_hyperparameters_1 = []
mean_scores_1 = []

for i in range (5):
    X_train, X_test, y_train, y_test = train_test_split(X_1, Y_1, train_size =␣
↪5000)

            # Create a pipeline - RF is a stand in, we will populate the␣
↪classifier part below
    pipe_svm = Pipeline([('std', StandardScaler()),
                 ('classifier', SVC())])

    search_space_svm = [{'classifier': [SVC()],
                 'classifier__kernel': ['rbf'],
                 'classifier__gamma': [.001, .005, .01, .05, .1, .5, 1, 2],
                 'classifier__C': [10^(-7), 10^(-6), 10^(-5), 10^(-4), 10^(-3),␣
↪10^(-2), 10^(-1), 10^(1), 10^(2), 10^(3)]},

                {'classifier': [SVC()],
                 'classifier__kernel': ['poly'],
                 'classifier__degree': [2, 3],
                 'classifier__C': [10^(-7), 10^(-6), 10^(-5), 10^(-4), 10^(-3),␣
↪10^(-2), 10^(-1), 10^(1), 10^(2), 10^(3)]},
```

```
                {'classifier': [SVC()],
                 'classifier__kernel': ['linear'],
                 'classifier__C': [10^(-7), 10^(-6), 10^(-5), 10^(-4), 10^(-3),␣
↪10^(-2), 10^(-1), 10^(1), 10^(2), 10^(3)]}
                ]

            # Create grid search
    clf_svm = GridSearchCV(pipe_svm, search_space_svm,␣
↪cv=StratifiedKFold(n_splits=5),
                   scoring=['accuracy', 'roc_auc', 'f1_micro'], refit=False,
                   verbose=0)

            # Fit grid search
    best_model_SVM = clf_svm.fit(X_train, y_train)

    h1 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_1.append(h1)
    optimal_hyperparameters_1.append(h2)
    optimal_hyperparameters_1.append(h3)
    mean_scores_1.append(best_model_SVM.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_accuracy'])])
    mean_scores_1.append(best_model_SVM.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_roc_auc'])])
    mean_scores_1.append(best_model_SVM.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_1)
print(mean_scores_1)
```

```
[{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 9,
'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
```

```
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'}]
[0.9917999999999999, 0.9973695290886383, 0.9917999999999999, 0.9926,
0.9965276692459965, 0.9926, 0.9938, 0.99797560483662, 0.9938, 0.9936,
0.9982742930603237, 0.9936, 0.9914, 0.9971614845611112, 0.9914]
```

0.3 SVM

letter dataset 2

[4]:
```python
optimal_hyperparameters_2 = []
mean_scores_2 = []

for i in range (5):
    X_train, X_test, y_train, y_test = train_test_split(X_2, Y_2, train_size =␣
↪5000)

    best_model_SVM = clf_svm.fit(X_train, y_train)

    h1 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_2.append(h1)
    optimal_hyperparameters_2.append(h2)
    optimal_hyperparameters_2.append(h3)
    mean_scores_2.append(best_model_SVM.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_accuracy'])])
    mean_scores_2.append(best_model_SVM.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_roc_auc'])])
    mean_scores_2.append(best_model_SVM.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_2)
print(mean_scores_2)
```

```
[{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.5,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'},
```

```
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.5,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.5,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.5,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 8,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.5,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.5, 'classifier__kernel': 'rbf'}]
[0.9574, 0.9915589324101474, 0.9574, 0.9564, 0.9914227406138234, 0.9564,
0.9541999999999999, 0.9906445443568952, 0.9541999999999999, 0.954,
0.9911424653113061, 0.954, 0.9543999999999999, 0.9908261666555502,
0.9543999999999999]
```

0.4 SVM

Adult Dataset

[5]:
```python
adult_main = pd.read_csv('adult.data')
adult_test = pd.read_csv ('adult.test')
adult = pd.concat([adult_main,adult_test], ignore_index = True)
#make class column binary
adult_ = adult.replace(' >50K', -1)
adult = adult_.replace(' <=50K', +1)

#move class column to be in 0th index
adult = adult[[ ' <=50K','39', ' State-gov', ' 77516', ' Bachelors', ' 13', '␣
↪Never-married',
       ' Adm-clerical', ' Not-in-family', ' White', ' Male', ' 2174', ' 0',
       ' 40', ' United-States', '|1x3 Cross validator']]

#rename columns according to data write-up
adult = adult.rename(columns={" <=50K": "Y", '39': 'age', ' State-gov':␣
↪'workclass', ' 77516': 'fnlwgt',
                              ' Bachelors': 'education', ' 13':␣
↪'education-num', ' Never-married': 'marital-status',
                              ' Adm-clerical':'occupation', ' Not-in-family' :␣
↪'relationship', ' White' : 'race',
                              ' Male' : 'sex', ' 2174' : 'capital-gain', ' 0' :␣
↪'capital-loss',' 40': 'hours-per-week',
                              ' United-States' : 'native-country'})
```

```
#one hot encode via pd.get_dummies
work = pd.get_dummies(adult['workclass'])
work.rename(columns = {' ?': 'NA_Work'}, inplace = True)

education = pd.get_dummies(adult['education'])
marital = pd.get_dummies(adult['marital-status'])
occu = pd.get_dummies(adult['occupation'])
occu.rename(columns = {' ?': 'NA_Occu'}, inplace = True)

relationship = pd.get_dummies(adult['relationship'])
race = pd.get_dummies(adult['race'])
sex = pd.get_dummies(adult['sex'])
country = pd.get_dummies(adult['native-country'])
country.rename(columns = {' ?': 'NA_Country'}, inplace = True)\

adult_= pd.concat([work,education,marital,occu,relationship,race,sex,country],␣
↪axis = 1)

#combine the continuous and categorical (now one hot encoded variables)
adult_ = pd.concat([adult[['Y',␣
↪'age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']],adult_],␣
↪axis =1)

adult_ = adult_[adult_.NA_Work == 0]
adult_ = adult_[adult_.NA_Occu == 0]
adult_ = adult_[adult_.NA_Country == 0]

adult_.drop(['NA_Work','NA_Occu','NA_Country'],axis = 1, inplace=True)

adult_.reset_index(inplace = True)
adult_.dropna(inplace = True)

#split off class data into seperate grouping
X_3 = adult_
Y_3 = X_3[['Y']]
X_3 = X_3.iloc[:, 1:]
```

[6]:
```
optimal_hyperparameters_3 = []
mean_scores_3 = []

for i in range (5):
    X_train, X_test, y_train, y_test = train_test_split(X_3, Y_3, train_size =␣
↪5000)

    best_model_SVM = clf_svm.fit(X_train, y_train)
```

```
    h1 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_3.append(h1)
    optimal_hyperparameters_3.append(h2)
    optimal_hyperparameters_3.append(h3)
    mean_scores_3.append(best_model_SVM.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_accuracy'])])
    mean_scores_3.append(best_model_SVM.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_roc_auc'])])
    mean_scores_3.append(best_model_SVM.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_SVM.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_3)
print(mean_scores_3)
```

```
[{'classifier': SVC(), 'classifier__C': 11, 'classifier__kernel': 'linear'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__degree': 3,
'classifier__kernel': 'poly'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__kernel': 'linear'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__kernel': 'linear'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__degree': 3, 'classifier__kernel': 'poly'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__kernel': 'linear'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__kernel': 'linear'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__degree': 3, 'classifier__kernel': 'poly'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__kernel': 'linear'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__kernel': 'linear'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__degree': 3,
'classifier__kernel': 'poly'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__kernel': 'linear'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__kernel': 'linear'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__degree': 3, 'classifier__kernel': 'poly'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__kernel': 'linear'}]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

0.5 SVM

Cov-type data

[7]:
```
#read in data move the class column to the 0th column
#rename columns (last 44 columns = leave one out arrangement)
cov_type = pd.read_csv('covtype.data')
cov_type = cov_type[['5', '2596', '51', '3', '258', '0', '510', '221', '232',␣
↪'148', '6279', '1',
```

```
        '0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9', '0.10',
        '0.11', '0.12', '0.13', '0.14', '0.15', '0.16', '0.17', '0.18', '0.19',
        '0.20', '0.21', '0.22', '0.23', '0.24', '0.25', '0.26', '0.27', '0.28',
        '0.29', '0.30', '0.31', '1.1', '0.32', '0.33', '0.34', '0.35', '0.36',
        '0.37', '0.38', '0.39', '0.40', '0.41', '0.42']]
cov_type.rename(columns = {'5':'Y', '2596':'Elevation', '51': 'Aspect', '3':␣
 ↪'Slope', '258':
                              'Horizontal_Distance_To_Hydrology', '0':␣
 ↪'Vertical_Distance_To_Hydrology', '510':
                              'Horizontal_Distance_To_Roadways', '221' :␣
 ↪'Hillshade_9am', '232' : 'Hillshade_Noon',
                              '148': 'Hillshade_3pm ', '6279':␣
 ↪'Horizontal_Distance_To_Fire_Points'}, inplace = True)

#make the classes binary so the largest class (2) is +1, rest is -1
cov_type = cov_type.replace(to_replace = {'Y': {1: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {3: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {4: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {5: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {6: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {7: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {2: +1}})

X_4 = cov_type
Y_4 = X_4[['Y']]
X_4 = X_4.iloc[:, 1:]
```

[8]:
```
optimal_hyperparameters_4 = []
mean_scores_4 = []

for i in range (5):
    X_train, X_test, y_train, y_test = train_test_split(X_4, Y_4, train_size =␣
 ↪5000)

    best_model_SVM = clf_svm.fit(X_train, y_train)

    h1 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
 ↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
 ↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_SVM.cv_results_['params'][ np.argmin(best_model_SVM.
 ↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_4.append(h1)
    optimal_hyperparameters_4.append(h2)
    optimal_hyperparameters_4.append(h3)
```

```
    mean_scores_4.append(best_model_SVM.cv_results_['mean_test_accuracy'][np.
 ↪argmin(best_model_SVM.cv_results_['rank_test_accuracy'])])
    mean_scores_4.append(best_model_SVM.cv_results_['mean_test_roc_auc'][np.
 ↪argmin(best_model_SVM.cv_results_['rank_test_roc_auc'])])
    mean_scores_4.append(best_model_SVM.cv_results_['mean_test_f1_micro'][np.
 ↪argmin(best_model_SVM.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_4)
print(mean_scores_4)
```

```
[{'classifier': SVC(), 'classifier__C': 8, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 8,
'classifier__gamma': 0.05, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 8, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 9,
'classifier__gamma': 0.05, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.1,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 9,
'classifier__gamma': 0.05, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.1, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 8, 'classifier__gamma': 0.05,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 8,
'classifier__gamma': 0.05, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 8, 'classifier__gamma': 0.05, 'classifier__kernel': 'rbf'},
{'classifier': SVC(), 'classifier__C': 11, 'classifier__gamma': 0.01,
'classifier__kernel': 'rbf'}, {'classifier': SVC(), 'classifier__C': 11,
'classifier__gamma': 0.01, 'classifier__kernel': 'rbf'}, {'classifier': SVC(),
'classifier__C': 11, 'classifier__gamma': 0.01, 'classifier__kernel': 'rbf'}]
[0.7896000000000001, 0.854787838721472, 0.7896000000000001, 0.7828,
0.8561640994828764, 0.7828, 0.798, 0.8658867285916727, 0.7980000000000002,
0.7976, 0.8629003681196661, 0.7976, 0.7896000000000001, 0.855717528179633,
0.7896000000000001]
```

[9]:
```
print(mean_scores_1)
print(mean_scores_2)
print(mean_scores_3)
print(mean_scores_4)
```

```
[0.9917999999999999, 0.9973695290886383, 0.9917999999999999, 0.9926,
0.9965276692459965, 0.9926, 0.9938, 0.99797560483662, 0.9938, 0.9936,
0.9982742930603237, 0.9936, 0.9914, 0.9971614845611112, 0.9914]
[0.9574, 0.9915589324101474, 0.9574, 0.9564, 0.9914227406138234, 0.9564,
0.9541999999999999, 0.9906445443568952, 0.9541999999999999, 0.954,
0.9911424653113061, 0.954, 0.9543999999999999, 0.9908261666555502,
0.9543999999999999]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

```
[0.7896000000000001, 0.854787838721472, 0.7896000000000001, 0.7828,
0.8561640994828764, 0.7828, 0.798, 0.8658867285916727, 0.7980000000000002,
0.7976, 0.8629003681196661, 0.7976, 0.7896000000000001, 0.855717528179633,
0.7896000000000001]
```

[ ]:

```

```

KNN

March 17, 2021

[1]:
```
%config InlineBackend.figure_format = 'retina'

from sklearn import datasets
import scipy
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set_style('white')  # plot formatting
from sklearn.pipeline import make_pipeline

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import validation_curve
from sklearn.metrics import r2_score
from sklearn.model_selection import learning_curve


from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

1 KNN

letter dataset 1

[2]:
```
#import letter data set
#letter1 for the problem with 'O' as the postive class rest as negative
#letter2, for the problem with A-M as positive and N-Z as negative
letter = pd.read_csv('letter-recognition.data')
```

```
letter_ = letter.replace('O', +1)
letter1 = letter_.replace(to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',␣
↪'I', 'J', 'K', 'L', 'M', 'N',
                                           'P', 'Q', 'R', 'S', 'T', 'U', 'V',␣
↪'W', 'X', 'Y', 'Z'], value = -1)
letter_Y = letter.replace (to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G',␣
↪'H', 'I', 'J', 'K', 'L', 'M'], value = +1)
letter2 = letter_Y.replace(to_replace = ['N', 'O', 'P', 'Q', 'R', 'S', 'T',␣
↪'U', 'V', 'W', 'X', 'Y', 'Z'], value = -1)

X_1 = letter1
Y_1 = X_1[['T']]
X_1 = X_1.iloc[:, 1:]

X_2 = letter2
Y_2 = X_2[['T']]
X_2 = X_2.iloc[:, 1:]
```

[3]:

```
optimal_hyperparameters_1 = []
mean_scores_1 = []

for i in range (5):
    X_train, X_test, y_train, y_test = train_test_split(X_1, Y_1, train_size =␣
↪5000)

            # Create a pipeline - RF is a stand in, we will populate the␣
↪classifier part below
    pipe_KNN = Pipeline([('std', StandardScaler()),
                 ('classifier', KNeighborsClassifier())])

    search_space_KNN = [{'classifier': [ KNeighborsClassifier()],
                        'classifier__n_neighbors' : [5, 10, 20, 200, 1000],
                        'classifier__weights' : ['uniform', 'distance'],
                        'classifier__metric' : ['euclidean', 'manhattan']}]

    clf_KNN = GridSearchCV(pipe_KNN, search_space_KNN,␣
↪cv=StratifiedKFold(n_splits=5),
                   scoring=['accuracy', 'roc_auc', 'f1_micro'], refit=False,
                   verbose=0)

            # Fit grid search
    best_model_KNN = clf_KNN.fit(X_train, y_train)

    h1 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_accuracy']) ]
```

```
    h2 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_1.append(h1)
    optimal_hyperparameters_1.append(h2)
    optimal_hyperparameters_1.append(h3)
    mean_scores_1.append(best_model_KNN.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_accuracy'])])
    mean_scores_1.append(best_model_KNN.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_roc_auc'])])
    mean_scores_1.append(best_model_KNN.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_1)
print(mean_scores_1)
```

```
[{'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'uniform'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'uniform'}]
```

```
[0.9878, 0.9953523361418097, 0.9878, 0.9898, 0.9874081443197305, 0.9898, 0.9904,
0.9940702748089011, 0.9904, 0.9895999999999999, 0.9956245103139947,
0.9895999999999999, 0.9902, 0.9930093776641092, 0.9902]
```

1.1 KNN

letter dataset 2

[4]:
```
optimal_hyperparameters_2 = []
mean_scores_2 = []

for i in range (5):
    pipe_KNN = Pipeline([('std', StandardScaler()),
                 ('classifier', KNeighborsClassifier())])

    search_space_KNN = [{'classifier': [ KNeighborsClassifier()],
                        'classifier__n_neighbors' : [5, 10, 20, 200, 1000],
                        'classifier__weights' : ['uniform', 'distance'],
                        'classifier__metric' : ['euclidean', 'manhattan']}]

    clf_KNN = GridSearchCV(pipe_KNN, search_space_KNN,␣
↪cv=StratifiedKFold(n_splits=5),
                   scoring=['accuracy', 'roc_auc', 'f1_micro'], refit=False,
                   verbose=0)


    X_train, X_test, y_train, y_test = train_test_split(X_2, Y_2, train_size =␣
↪5000)

    best_model_KNN = clf_KNN.fit(X_train, y_train)

    h1 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_2.append(h1)
    optimal_hyperparameters_2.append(h2)
    optimal_hyperparameters_2.append(h3)
    mean_scores_2.append(best_model_KNN.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_accuracy'])])
    mean_scores_2.append(best_model_KNN.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_roc_auc'])])
    mean_scores_2.append(best_model_KNN.cv_results_['mean_test_f1_micro'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_f1_micro'])])
```

```python
print(optimal_hyperparameters_2)
print(mean_scores_2)
```

```
[{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}]
[0.9442, 0.9901782461359016, 0.9442, 0.942, 0.9868975012500636, 0.942, 0.9458,
0.9874528258141779, 0.9458, 0.9438000000000001, 0.9888179770245561,
0.9438000000000001, 0.945, 0.9881637738226587, 0.945]
```

1.2 KNN

Adult dataset

[2]:
```python
adult_main = pd.read_csv('adult.data')
adult_test = pd.read_csv ('adult.test')
adult = pd.concat([adult_main,adult_test], ignore_index = True)
#make class column binary
adult_ = adult.replace(' >50K', -1)
adult = adult_.replace(' <=50K', +1)
```

```
#move class column to be in 0th index
adult = adult[[ ' <=50K','39', ' State-gov', ' 77516', ' Bachelors', ' 13', ' Never-married',
       ' Adm-clerical', ' Not-in-family', ' White', ' Male', ' 2174', ' 0',
       ' 40', ' United-States', '|1x3 Cross validator']]

#rename columns according to data write-up
adult = adult.rename(columns={" <=50K": "Y", '39': 'age', ' State-gov': 'workclass', ' 77516': 'fnlwgt',
                              ' Bachelors': 'education', ' 13': 'education-num', ' Never-married': 'marital-status',
                              ' Adm-clerical':'occupation', ' Not-in-family' : 'relationship', ' White' : 'race',
                              ' Male' : 'sex', ' 2174' : 'capital-gain', ' 0' : 'capital-loss',' 40': 'hours-per-week',
                              ' United-States' : 'native-country'})

#one hot encode via pd.get_dummies
work = pd.get_dummies(adult['workclass'])
work.rename(columns = {' ?': 'NA_Work'}, inplace = True)

education = pd.get_dummies(adult['education'])
marital = pd.get_dummies(adult['marital-status'])
occu = pd.get_dummies(adult['occupation'])
occu.rename(columns = {' ?': 'NA_Occu'}, inplace = True)

relationship = pd.get_dummies(adult['relationship'])
race = pd.get_dummies(adult['race'])
sex = pd.get_dummies(adult['sex'])
country = pd.get_dummies(adult['native-country'])
country.rename(columns = {' ?': 'NA_Country'}, inplace = True)\

adult_= pd.concat([work,education,marital,occu,relationship,race,sex,country], axis = 1)

#combine the continuous and categorical (now one hot encoded variables)
adult_ = pd.concat([adult[['Y', 'age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']],adult_], axis =1)

adult_ = adult_[adult_.NA_Work == 0]
adult_ = adult_[adult_.NA_Occu == 0]
adult_ = adult_[adult_.NA_Country == 0]

adult_.drop(['NA_Work','NA_Occu','NA_Country'],axis = 1, inplace=True)
```

```
adult_.reset_index(inplace = True)
adult_.dropna(inplace = True)

#split off class data into seperate grouping
X_3 = adult_
Y_3 = X_3[['Y']]
X_3 = X_3.iloc[:, 1:]
```

[3]:
```
optimal_hyperparameters_3 = []
mean_scores_3 = []

for i in range (5):
    pipe_KNN = Pipeline([('std', StandardScaler()),
                ('classifier', KNeighborsClassifier())])

    search_space_KNN = [{'classifier': [ KNeighborsClassifier()],
                        'classifier__n_neighbors' : [5, 10, 20, 200, 1000],
                        'classifier__weights' : ['uniform', 'distance'],
                        'classifier__metric' : ['euclidean', 'manhattan']}]

    clf_KNN = GridSearchCV(pipe_KNN, search_space_KNN,␣
↪cv=StratifiedKFold(n_splits=5),
                    scoring=['accuracy', 'roc_auc', 'f1_micro'], refit=False,
                    verbose=0)

            # Fit grid search
    X_train, X_test, y_train, y_test = train_test_split(X_3, Y_3, train_size =␣
↪5000)

    best_model_KNN = clf_KNN.fit(X_train, y_train)

    h1 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_3.append(h1)
    optimal_hyperparameters_3.append(h2)
    optimal_hyperparameters_3.append(h3)
    mean_scores_3.append(best_model_KNN.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_accuracy'])])
    mean_scores_3.append(best_model_KNN.cv_results_['mean_test_roc_auc'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_roc_auc'])])
```

```
    mean_scores_3.append(best_model_KNN.cv_results_['mean_test_f1_micro'][np.
 ↪argmin(best_model_KNN.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_3)
print(mean_scores_3)
```

```
[{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 200, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 200, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 200, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 200, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 20, 'classifier__weights': 'distance'}]
[0.9438000000000001, 0.9817093088027209, 0.9438000000000001, 0.9457999999999999,
0.9818624835471507, 0.9457999999999999, 0.9492, 0.9816082949566619, 0.9492,
0.9495999999999999, 0.9829321909686362, 0.9495999999999999, 0.9454,
0.9817632257340396, 0.9454]
```

[4]:

```
#read in data move the class column to the 0th column
#rename columns (last 44 columns = leave one out arrangement)
cov_type = pd.read_csv('covtype.data')
cov_type = cov_type[['5', '2596', '51', '3', '258', '0', '510', '221', '232',␣
 ↪'148', '6279', '1',
       '0.1', '0.2', '0.3', '0.4', '0.5', '0.6', '0.7', '0.8', '0.9', '0.10',
       '0.11', '0.12', '0.13', '0.14', '0.15', '0.16', '0.17', '0.18', '0.19',
```

```
      '0.20', '0.21', '0.22', '0.23', '0.24', '0.25', '0.26', '0.27', '0.28',
      '0.29', '0.30', '0.31', '1.1', '0.32', '0.33', '0.34', '0.35', '0.36',
      '0.37', '0.38', '0.39', '0.40', '0.41', '0.42']]
cov_type.rename(columns = {'5':'Y', '2596':'Elevation', '51': 'Aspect', '3':␣
↪'Slope', '258':
                          'Horizontal_Distance_To_Hydrology', '0':␣
↪'Vertical_Distance_To_Hydrology', '510':
                          'Horizontal_Distance_To_Roadways', '221' :␣
↪'Hillshade_9am', '232' : 'Hillshade_Noon',
                          '148': 'Hillshade_3pm ', '6279':␣
↪'Horizontal_Distance_To_Fire_Points'}, inplace = True)

#make the classes binary so the largest class (2) is +1, rest is -1
cov_type = cov_type.replace(to_replace = {'Y': {1: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {3: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {4: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {5: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {6: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {7: -1}})
cov_type = cov_type.replace(to_replace = {'Y': {2: +1}})

X_4 = cov_type
Y_4 = X_4[['Y']]
X_4 = X_4.iloc[:, 1:]
```

[5]:
```
optimal_hyperparameters_4 = []
mean_scores_4 = []

for i in range (5):
    X_train, X_test, y_train, y_test = train_test_split(X_4, Y_4, train_size =␣
↪5000)

    best_model_KNN = clf_KNN.fit(X_train, y_train)

    h1 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_accuracy']) ]
    h2 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_roc_auc']) ]
    h3 = best_model_KNN.cv_results_['params'][ np.argmin(best_model_KNN.
↪cv_results_['rank_test_f1_micro']) ]

    optimal_hyperparameters_4.append(h1)
    optimal_hyperparameters_4.append(h2)
    optimal_hyperparameters_4.append(h3)
    mean_scores_4.append(best_model_KNN.cv_results_['mean_test_accuracy'][np.
↪argmin(best_model_KNN.cv_results_['rank_test_accuracy'])])
```

```
    mean_scores_4.append(best_model_KNN.cv_results_['mean_test_roc_auc'][np.
 ↪argmin(best_model_KNN.cv_results_['rank_test_roc_auc'])])
    mean_scores_4.append(best_model_KNN.cv_results_['mean_test_f1_micro'][np.
 ↪argmin(best_model_KNN.cv_results_['rank_test_f1_micro'])])
print(optimal_hyperparameters_4)
print(mean_scores_4)
```

```
[{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}, {'classifier':
KNeighborsClassifier(), 'classifier__metric': 'manhattan',
'classifier__n_neighbors': 10, 'classifier__weights': 'distance'},
{'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean',
'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}]
[0.7714, 0.8486835497479908, 0.7714000000000001, 0.7796000000000001,
0.8566611340063772, 0.7796000000000001, 0.7756, 0.8527088740918923, 0.7756,
0.7794, 0.8586781335696202, 0.7794, 0.7844, 0.8639497428160146,
0.7844000000000001]
```

[ ]: