

## mCLESS Final Project

### Introduction:

This project implements the Multi-Class Least Error Square Sum (mCLESS) family of interpretable machine learning algorithms. It solves a multi-column least-squares minimization problem

$$\hat{W} = \arg \min_W \|AW - B\|^2,$$

where  $A = [1, x_1, x_2, \dots, x_d]$  represents columns of the training dataset and  $B$  is an incidence matrix that corresponds to the labels of the training data, in order to derive a weight matrix  $\hat{W}$  that may be used for predictions on data with the same dimensionality. To avoid rank-deficiency issues in the LS problem, a singular value decomposition  $A = U\Sigma V^T$  is used in place of  $A$  in the formulation above. The LS problem is then solved for a minimum-norm solution using the pseudoinverse as

$$W = A^+B = V\Sigma^{-1}U^TB.$$

The model is tested on 3 datasets from scikit-learn, the Iris dataset, the Wine dataset, and the Breast Cancer dataset. Additionally, each dataset is tested using various feature expansions in an attempt to improve performance. All variations of the model, as well as some other common classifiers, are tested with and without data normalization to get a sense of the impact of problems related to data scaling on particular models. The measurement of performance is strictly focused on accuracy, not time or memory efficiency.

### Modules:

*All code below located in Appendix*

mcless.py

The mCLESS module is very simple and not fully-featured since the focus on the project rubric seemed to put a much bigger emphasis on feature expansion as a test of accuracy improvement. As a result, the module simply solves the LS problem for the weight matrix using training data and returns a prediction when given test data.

preprocessing\_util.py

A simple set of functions were implemented in this module to perform basic preprocessing steps such as data normalization and imputation in the case of nan values or infinity. It has not been fully realized and contains very little functionality, but will be fleshed out independently after submission of this project.

feature\_expansion.py

The feature expansion module is a collection of functions that take some data and appends new features (as column vectors) to the end of the matrix. Specifically, each sample of the data (each row) has one or more values added to the end based on some vector  $p$  that is calculated to find the feature expansion based on the Euclidean distance

$$\phi(x) = ||x - p||_2.$$

One of the simplest feature expansions this project used found the mean of each column, or mean of each feature among all given samples, and constructed a vector  $p = [\bar{x}_1, \bar{x}_2, \dots, \bar{x}_d]$  to be used in the Euclidean distance. This added a single new feature,  $x_{d+1}$ . Another feature expansion used the same concept, but instead formed  $p$  using the column medians. This also added a single new feature.

Another method attempted to use a concept from Gaussian kernels, which were foregone in this project, instead using the Taylor expansion of the Gaussian kernel, which for any parameter  $\sigma > 0$  gives a feature expansion such that

$$\phi(x)^T \phi(y) = e^{-\frac{||x-y||_2^2}{2\sigma^2}}.$$

In this case, the project simply used the Taylor expansion of  $\phi(x)$ ,

$$\phi(x) = e^{-\frac{x^2}{2\sigma^2}} \left[ 1, \dots, \frac{1}{j!} \left( \frac{x}{\sigma} \right)^j, \dots, \frac{1}{n!} \left( \frac{x}{\sigma} \right)^n \right] \text{ for } j = 0, 1, \dots, n$$

for a vector of  $n+1$  dimensions. In this case, two implementations were created. The single Taylor expansion used the single vector created above (with  $n=d-1$ ) as  $p$  so that a single new feature  $\phi(x) = ||x - p||_2$  is added, while the multi-Taylor expansion used  $d$  vectors, where for each element in the vector that would be created for the single Taylor expansion, a vector of  $d$  identical elements is used. Thus the  $d$  new features are  $||x - p_1||_2, ||x - p_2||_2, \dots, ||x - p_d||_2$ .

Finally, a more analytical approach that ultimately showed some logical issues was choosing  $p$  as a direction of steepest descent that minimized a loss function. In this case, the gradient of the MSE (Mean Square Error) of the LS problem was used. Thus, this required the use of the weight matrix, pre-trained on the original dataset. Also, because of the dimensionality requirement of the test data, it is trained using the test labels, which totally invalidates the test. It remains in the project because I believe that this approach of choosing  $p$  as a descent direction of a loss function is ultimately the best course of action. A smarter choice of loss function that is dependent only on the training data, along with implementation of a line search, would likely yield great results for a feature expansion. I consider this to be necessary future research.

Below are the results for all 3 datasets, with and without normalization, between 9 different classification tests including 3 different models besides mCLESS.

Output:

MCLESS testing on sklearn Iris dataset

```
#####  
.. _iris_dataset:
```

Iris plants dataset

-----

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 150 (50 in each of three classes)

:Number of Attributes: 4 numeric, predictive attributes and the class

:Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

:Missing Attribute Values: None

:Class Distribution: 33.3% for each of 3 classes.

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the

pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

.. topic:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

#####

**Iris dataset features:**

**['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']**

**Iris dataset class labels:**

**['setosa' 'versicolor' 'virginica']**

**number of data samples: 150**

**number of features per sample: 4**

**Progress on Iris dataset: 100%|██████████| 10/10 [00:01<00:00, 5.55it/s]**

**mean accuracies for the test predictions of unnormed data:**

**MCLESS w/ original : 0.8244444444444445**

**MCLESS w/ mse : 0.9955555555555555**

**MCLESS w/ single taylor : 0.8466666666666667**

**MCLESS w/ multi taylor : 0.96**

**MCLESS w/ col mean : 0.9622222222222222**

**MCLESS w/ col median : 0.9511111111111111**

**Naive Bayes w/ original : 0.9488888888888889**  
**K Neighbors w/ original : 0.9577777777777777**  
**Random Forest w/ original : 0.9466666666666667**

**mean accuracies for the test predictions of normed data:**

**MCLESS w/ original : 0.8266666666666668**  
**MCLESS w/ mse : 0.9400000000000001**  
**MCLESS w/ single taylor : 0.8200000000000001**  
**MCLESS w/ multi taylor : 0.9377777777777778**  
**MCLESS w/ col mean : 0.9355555555555556**  
**MCLESS w/ col median : 0.9222222222222223**  
**Naive Bayes w/ original : 0.9355555555555556**  
**K Neighbors w/ original : 0.9377777777777778**  
**Random Forest w/ original : 0.9444444444444444**

MCLESS testing on sklearn Wine dataset

#####  
.. \_wine\_dataset:

Wine recognition dataset

-----

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 178  
:Number of Attributes: 13 numeric, predictive attributes and the class  
:Attribute Information:  
    - Alcohol  
    - Malic acid  
    - Ash  
    - Alcalinity of ash  
    - Magnesium  
    - Total phenols  
    - Flavanoids  
    - Nonflavanoid phenols  
    - Proanthocyanins  
    - Color intensity  
    - Hue  
    - OD280/OD315 of diluted wines  
    - Proline  
  
- class:  
    - class\_0  
    - class\_1  
    - class\_2

:Summary Statistics:

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

:Missing Attribute Values: None  
:Class Distribution: class\_0 (59), class\_1 (71), class\_2 (48)  
:Creator: R.A. Fisher  
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)  
:Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.  
<https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

Original Owners:

Forina, M. et al, PARVUS -  
An Extendible Package for Data Exploration, Classification and Correlation.  
Institute of Pharmaceutical and Food Analysis and Technologies,  
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository  
[<https://archive.ics.uci.edu/ml>]. Irvine, CA: University of California,  
School of Information and Computer Science.

.. topic:: References

(1) S. Aeberhard, D. Coomans and O. de Vel,  
Comparison of Classifiers in High Dimensional Settings,  
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of  
Mathematics and Statistics, James Cook University of North Queensland.  
(Also submitted to Technometrics).

The data was used with many others for comparing various  
classifiers. The classes are separable, though only RDA  
has achieved 100% correct classification.  
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))  
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,  
"THE CLASSIFICATION PERFORMANCE OF RDA"  
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of  
Mathematics and Statistics, James Cook University of North Queensland.  
(Also submitted to Journal of Chemometrics).

#####

**Wine dataset features:**

['alcohol', 'malic\_acid', 'ash', 'alkalinity\_of\_ash', 'magnesium', 'total\_phenols', 'flavanoids',  
'nonflavanoid\_phenols', 'proanthocyanins', 'color\_intensity', 'hue', 'od280/od315\_of\_diluted\_wines',  
'proline']

**Wine dataset class labels:**

['class\_0' 'class\_1' 'class\_2']

**number of data samples: 178**

**number of features per sample: 13**

**Progress on Wine dataset: 100%**  **10/10 [00:02<00:00, 3.37it/s]**

**mean accuracies for the test predictions of unnormed data:**

**MCLESS w/ original : 0.9814814814814815**

**MCLESS w/ mse : 0.8388888888888889**

**MCLESS w/ single taylor : 0.9777777777777776**

**MCLESS w/ multi taylor : 0.9833333333333334**

**MCLESS w/ col mean : 0.9796296296296296**

**MCLESS w/ col median : 0.9777777777777776**

**Naive Bayes w/ original : 0.9740740740740741**

**K Neighbors w/ original : 0.6944444444444445**

**Random Forest w/ original : 0.9851851851851852**

**mean accuracies for the test predictions of normed data:**

**MCLESS w/ original : 0.9851851851851852**  
**MCLESS w/ mse : 0.7833333333333333**  
**MCLESS w/ single taylor : 0.987037037037037**  
**MCLESS w/ multi taylor : 0.975925925925926**  
**MCLESS w/ col mean : 0.9851851851851852**  
**MCLESS w/ col median : 0.9833333333333334**  
**Naive Bayes w/ original : 0.9777777777777776**  
**K Neighbors w/ original : 0.962962962962963**  
**Random Forest w/ original : 0.9851851851851853**

MCLESS testing on sklearn Breast Cancer dataset

```
#####  
.. _breast_cancer_dataset:
```

Breast cancer wisconsin (diagnostic) dataset

-----

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness ( $\text{perimeter}^2 / \text{area} - 1.0$ )
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three worst/largest values) of these features were computed for each image, resulting in 30 features. For instance, field 0 is Mean Radius,

field

10 is Radius SE, field 20 is Worst Radius.

- class:

- WDBC-Malignant
- WDBC-Benign

:Summary Statistics:



	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.  
<https://goo.gl/U2Uwz2>

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

.. topic:: References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

#####

**Breast Cancer dataset features:**

**['mean radius' 'mean texture' 'mean perimeter' 'mean area'**  
**'mean smoothness' 'mean compactness' 'mean concavity'**


'mean concave points' 'mean symmetry' 'mean fractal dimension'  
'radius error' 'texture error' 'perimeter error' 'area error'  
'smoothness error' 'compactness error' 'concavity error'  
'concave points error' 'symmetry error' 'fractal dimension error'  
'worst radius' 'worst texture' 'worst perimeter' 'worst area'  
'worst smoothness' 'worst compactness' 'worst concavity'  
'worst concave points' 'worst symmetry' 'worst fractal dimension']

**Breast Cancer dataset class labels:**

['malignant' 'benign']

**number of data samples: 569**

**number of features per sample: 30**

**Progress on Breast Cancer dataset: 100%  10/10 [00:12<00:00, 1.29s/it**

**mean accuracies for the test predictions of unnormed data:**

**MCLESS w/ original : 0.9578947368421054**

**MCLESS w/ mse : 0.9923976608187134**

**MCLESS w/ single taylor : 0.9590643274853802**

**MCLESS w/ multi taylor : 0.9532163742690057**

**MCLESS w/ col mean : 0.9619883040935674**

**MCLESS w/ col median : 0.9555555555555555**

**Naive Bayes w/ original : 0.9426900584795321**

**K Neighbors w/ original : 0.9333333333333332**

**Random Forest w/ original : 0.9485380116959063**

**mean accuracies for the test predictions of normed data:**

**MCLESS w/ original : 0.9567251461988304**

**MCLESS w/ mse : 0.9818713450292396**

**MCLESS w/ single taylor : 0.9514619883040935**

**MCLESS w/ multi taylor : 0.9584795321637427**

**MCLESS w/ col mean : 0.956140350877193**

**MCLESS w/ col median : 0.9573099415204679**

**Naive Bayes w/ original : 0.9368421052631579**

**K Neighbors w/ original : 0.9619883040935673**

**Random Forest w/ original : 0.9502923976608187**

main.py

```
import pandas as pd
import numpy as np
import seaborn as sbn
from copy import deepcopy
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris, load_wine, load_breast_cancer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

from mcless import mcless
import feature_expansion as FE
import preprocessing_util as PREP

def get_test_accuracy(prediction, y_test, test_size):
    num_correct = np.sum([np.argmax(prediction[k]) == y_test[k] for k in
range(test_size)])
    return num_correct/test_size

def test_feature_expansions(model, X_train, X_test, y_train, y_test):
    accuracy = np.zeros(5)
    test_size = len(y_test)
    mse_prediction = FE.test_mse_features(model, X_train, X_test, y_train, y_test)
    accuracy[0] = get_test_accuracy(mse_prediction, y_test, test_size)
    single_taylor_pred, multi_taylor_pred = FE.test_taylor_features(X_train, y_train,
X_test)
    accuracy[1] = get_test_accuracy(single_taylor_pred, y_test, test_size)
    accuracy[2] = get_test_accuracy(multi_taylor_pred, y_test, test_size)
    mean_feat_pred = FE.test_mean_features(X_train, y_train, X_test)
    accuracy[3] = get_test_accuracy(mean_feat_pred, y_test, test_size)
    med_feat_pred = FE.test_med_features(X_train, y_train, X_test)
    accuracy[4] = get_test_accuracy(med_feat_pred, y_test, test_size)
    return accuracy

def get_acc_dict(num_runs):
    return {'MCLESS w/ original' : np.zeros(num_runs),
            'MCLESS w/ mse' : np.zeros(num_runs),
            'MCLESS w/ single taylor' : np.zeros(num_runs),
            'MCLESS w/ multi taylor' : np.zeros(num_runs),
            'MCLESS w/ col mean' : np.zeros(num_runs),
```

```

        'MCLESS w/ col median' : np.zeros(num_runs),
        'Naive Bayes w/ original' : np.zeros(num_runs),
        'K Neighbors w/ original' : np.zeros(num_runs),
        'Random Forest w/ original' : np.zeros(num_runs)}

def test_model_predictions(X_train, X_test, y_train, y_test):
    accuracy = np.zeros(9)
    model = mcless(X_train, y_train)
    model.compute_training_matrices()
    prediction = model.predict(X_test)
    accuracy[0] = get_test_accuracy(prediction, y_test, test_size)
    accuracy[1:6] = test_feature_expansions(model, X_train, X_test, y_train, y_test)
    accuracy[6:] = test_sklearn_models(X_train, X_test, y_train, y_test)
    return accuracy

def test_sklearn_models(X_train, X_test, y_train, y_test):
    names = ['Naive Bayes', 'K Neighbors', 'Random Forest']
    classifiers = [GaussianNB(),
                   KNeighborsClassifier(7),
                   RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1)]
    accuracy = np.zeros(len(names))
    for clf, count in zip(names, classifiers, range(len(names))):
        clf.fit(X_train, y_train)
        accuracy[count] = clf.score(X_test, y_test)
    return accuracy

def print_mean_accuracy(acc_dict, header):
    print(header)
    for key, val in acc_dict.items():
        print(f'{key} : {np.mean(val)}')

def print_data_description(dataset, dataset_name, shape):
    print(f'MCLESS testing on sklearn {dataset_name} dataset')
    print('#####')
    print(dataset.DESCR, '\n')
    print('#####')
    print(f'{dataset_name} dataset features:\n{dataset.feature_names}\n')
    print(f'{dataset_name} dataset class labels:\n{dataset.target_names}\n')
    print(f'number of data samples: {shape[0]}\n')
    print(f'number of features per sample: {shape[1]}\n')

if __name__ == '__main__':
    datasets = [load_iris(), load_wine(), load_breast_cancer()]
    dataset_names = ['Iris', 'Wine', 'Breast Cancer'] # since not all have filenames
    apparently

```

```

num_runs = 10
num_datasets = len(datasets)
set_count = 0
for data_read in datasets:
    X = data_read.data
    y = data_read.target
    N, d = X.shape
    print_data_description(data_read, dataset_names[set_count], X.shape)
    label_set = set(y)
    num_classes = len(label_set)
    acc_dict = get_acc_dict(num_runs)
    acc_dict_norm = get_acc_dict(num_runs)
    for i in tqdm(range(num_runs), total=num_runs, desc=f'Progress on
{dataset_names[set_count]} dataset'):
        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,
random_state=i,stratify=y)
        PREP.fill_nan(X_train)
        PREP.fill_nan(X_test)
        test_size = len(y_test)
        X_train_norm = PREP.normalize_data(X_train, d)
        X_test_norm = PREP.normalize_data(X_test, d)

        unnormed_acc = test_model_predictions(X_train, X_test, y_train, y_test)
        normed_acc = test_model_predictions(X_train_norm, X_test_norm, y_train,
y_test)

        loop_count = 0
        for key in acc_dict.keys():
            acc_dict[key][i] = unnormed_acc[loop_count]
            acc_dict_norm[key][i] = normed_acc[loop_count]
            loop_count += 1
        unnormed_header = '\nmean accuracies for the test predictions of unnormed data:'
        normed_header = '\nmean accuracies for the test predictions of normed data:'
        print_mean_accuracy(acc_dict, unnormed_header)
        print_mean_accuracy(acc_dict_norm, normed_header)
    set_count += 1

```

mcless.py

```

import numpy as np

class mcless(object):
    def __init__(self, X, y):
        self.X = X
        self.labels = y
        self.N, self.d = X.shape

```

```

        self.num_classes = len(set(y))

def get_information_matrix(self):
    self.A = np.concatenate((np.ones((self.N,1)), self.X), axis=1)
    # print(f'shape of A: {self.A.shape}')

    '''# initialize weights
def get_weight_matrix(self):
    self.W = np.random.rand(self.d+1, self.num_classes)
    # print(f'shape of W: {self.W.shape}')'''

def get_source_matrix(self):
    self.B = np.zeros((self.N, self.num_classes))
    for i in range(self.N):
        self.B[i,self.labels[i]] = 1
    # print(f'shape of B: {self.B.shape}')

def get_SVD(self):
    U, Sigma, V_T = np.linalg.svd(self.A, full_matrices=False)
    self.SVD = (U, np.diag(Sigma), V_T)

def get_pseudoinverse(self):
    self.A_plus = self.SVD[2].T @ np.diag((1/np.diag(self.SVD[1]))) @ self.SVD[0].T
    # print(f'shape of A_plus: {self.A_plus.shape}')

def get_min_norm(self):
    self.W_hat = self.A_plus @ self.B
    # print(f'shape of W_hat: {self.W_hat.shape}')

def compute_training_matrices(self):
    self.get_information_matrix()
    self.get_source_matrix()
    self.get_SVD()
    self.get_pseudoinverse()
    self.get_min_norm()

def predict(self, X_test):
    m, n = X_test.shape
    A_test = np.concatenate((np.ones((m,1)), X_test), axis=1)
    return A_test @ self.W_hat

# TODO: check rank of A and possibly implement rank-deficient solution
# TODO: refer to page 153-154 of textbook for solving LS problems by partitioning
# TODO: implement feature selection within mcless - check PCA on pg 156

```

## preprocessing\_util.py

```
import numpy as np
from copy import deepcopy

def normalize_data(X, d):
    X_norm = deepcopy(X)
    for col in range(d):
        X_norm[:,col] -= np.mean(X_norm[:,col])
        X_norm[:,col] /= np.std(X_norm[:,col])
    return X_norm

def normalize_data2(X, d):
    X_norm = deepcopy(X)
    for col in range(d):
        X_norm[:,col] /= np.linalg.norm(X_norm[:,col])
    return X_norm

def fill_nan(X):
    for i in range(X.shape[1]):
        col_mean = np.mean(X[:,i])
        np.nan_to_num(X[:,i], copy=False, nan=col_mean)

# TODO: add method to fill nan values
# TODO: feature selection (PCA), possibly
# TODO: get rank of data matrix and check singularity
# implement more than one method to solve for the weight matrix
```

## feature\_expansion.py

```
import numpy as np
from mclass import mclass
from scipy.optimize import minimize

# Taylor expansion of Gaussian kernel
def get_Taylor(x, sigma):
    num_var = len(x)
    phi_x = np.zeros(num_var)
    mult = np.exp(-((x/sigma)**2)/2)
    for i in range(num_var):
        phi_x[i] = np.dot((1/np.math.factorial(i))*((x/sigma)**i), mult)
    return phi_x

def multi_Taylor_exp(X, N, d):
    X_new = np.zeros((N,d))
```



```

new_features = np.zeros((N,d))
for i in range(N):
    X_new[i,:] = get_Taylor(X[i,:], 0.25)
    # covers d different points p of the form [p_j, p_j, ..., p_j] for ||x-p||
    for j in range(d):
        new_features[i,j] = np.linalg.norm(X[i,:] - X_new[i,j])
return np.concatenate((X, new_features), axis=1)

def single_Taylor_exp(X, N, d):
    X_new = np.zeros((N,d))
    new_features = np.zeros((N,1))
    # covers a single point p for each sample so that the new feature is ||x-p||
    for i in range(N):
        X_new[i,:] = get_Taylor(X[i,:], 0.25)
        new_features[i,:] = np.linalg.norm(X[i,:] - X_new[i,:])
    return np.concatenate((X, new_features), axis=1)

def single_col_mean_exp(X, N, d):
    new_feature = np.zeros((N,1))
    column_mean = np.array([np.mean(X[:,i]) for i in range(d)])
    for i in range(N):
        new_feature[i] = np.linalg.norm(X[i,:] - column_mean)
    return np.concatenate((X, new_feature), axis=1)

def single_col_med_exp(X, N, d):
    new_feature = np.zeros((N,1))
    column_mean = np.array([np.median(X[:,i]) for i in range(d)])
    for i in range(N):
        new_feature[i] = np.linalg.norm(X[i,:] - column_mean)
    return np.concatenate((X, new_feature), axis=1)

# lots of issues with this logic, but time doesn't permit testing of a new loss function
def mse_loss_exp(A, y, W, N, d):
    # y: scalar, a: vector of size (d+1), w: vector of size (d+1)
    '''print(f'shape of A: {A.shape}')
```

```

    print(f'shape of W_hat: {W.shape}')
    print(f'shape of y: {y.shape}')
    print(f'N,d = {N,d}')'''
    grad = lambda a, w, y, a_i: (-2*a_i)*(y - np.dot(a,w))
    new_feature = np.zeros((N,len(W[0,:])))
    for k in range(len(W[0,:])):
        for i in range(N):
            p = np.array([grad(A[i,:], W[:,k], y[i], A[i,j]) for j in range(d)])
            new_feature[i,k] = np.linalg.norm(A[i,:] - p)
    return new_feature

```

```

def test_mse_features(model, X_train, X_test, y_train, y_test):
    N, d = X_test.shape
    X_new = np.concatenate((np.ones((N,1))), X_test), axis=1
    new_train_features = mse_loss_exp(model.A, y_train, model.W_hat, *model.A.shape)
    # I realize the logical issue with the line below - I'm leaving this method in either
way
    # to show that I understand p should probably be the maximum decrease in a loss
function
    new_test_features = mse_loss_exp(X_new, y_test, model.W_hat, N, d+1)
    X_train_new = np.concatenate((model.A, new_train_features), axis=1)
    X_test_new = np.concatenate((X_new, new_test_features), axis=1)
    fe_model = mcless(X_train_new, y_train)
    fe_model.compute_training_matrices()
    return fe_model.predict(X_test_new)

def test_taylor_features(X_train, y_train, X_test):
    X_train_single = single_Taylor_exp(X_train, *X_train.shape)
    X_train_multi = multi_Taylor_exp(X_train, *X_train.shape)
    X_test_single = single_Taylor_exp(X_test, *X_test.shape)
    X_test_multi = multi_Taylor_exp(X_test, *X_test.shape)
    single_model = mcless(X_train_single, y_train)
    single_model.compute_training_matrices()
    multi_model = mcless(X_train_multi, y_train)
    multi_model.compute_training_matrices()
    return single_model.predict(X_test_single), multi_model.predict(X_test_multi)

def test_mean_features(X_train, y_train, X_test):
    X_train_new = single_col_mean_exp(X_train, *X_train.shape)
    X_test_new = single_col_mean_exp(X_test, *X_test.shape)
    model = mcless(X_train_new, y_train)
    model.compute_training_matrices()
    return model.predict(X_test_new)

def test_med_features(X_train, y_train, X_test):
    X_train_new = single_col_med_exp(X_train, *X_train.shape)
    X_test_new = single_col_med_exp(X_test, *X_test.shape)
    model = mcless(X_train_new, y_train)
    model.compute_training_matrices()
    return model.predict(X_test_new)

```