EE 510: Machine Learning Theory & Algorithms **Spring 2022**

Mini Project 1

Due: April 22, 2022, 11:59PM PT

Student Name: Martín Rodriguez *Instructor Name: John Lipor*

1 Problem description

The Titanic dataset is billed as an "intro" dataset hosted by the popular data science competition website, Kaggle [1]. It contains data on the passengers on the famous Titanic steamliner, which sank on its maiden voyage in 1912. It has a training set and a testing set; the training set is used to train the model and the trained model is then used to make predictions about the testing set. Only the training set contains the true labels, which in this case is a binary value: 1 if the passenger survived and 0 if the passenger died. The objective is to create a linear model that can accurately predict whether a passenger lived or died using the available data.

Ridge regression aims to optimize the cost function given a set of sample-label pairs (x_i, y_i) for m samples, K classes, and d features:

$$\widehat{W} = \underset{W \in \mathbb{R}^{d \times K}}{\arg\min} \sum_{i=1}^{m} \left\| W^T x_i - y_i \right\|_2^2 + \lambda \left\| W \right\|_F^2$$

where the normal equation solved for $\widehat{W}$ is

$$\widehat{W} = (X^T X + \lambda I)^{-1} X^T y$$

2 Exploratory data analysis (EDA)

The dataset consists of two CSV (Comma-Separated Values) files which have already been separated into "train" and "test" subsets. The files contain categorical, numerical, and string-based data in addition to quite a few NaNs (missing values). In total, `train.csv` contains 891 data points with 12 features and `test.csv` contains 418 data points with 11 features. `test.csv` lacks the column for the "Survived" classification, which the model aims to predict). In order to validate the model, the data in `train.csv` was further separated into a test set and validation set. Of the 12 test features, 3 contain NaNs in at least one of the 891 data points: `['Age']`, `['Cabin']`, and `['Embarked']`, which amounts to a total of 708 data points with at least one missing feature. If all data points with NaNs are removed, this only leaves 183 points total for training and validation. Because this causes the number of data points to be reduced so heavily, it could be worthwhile to explore different methods of data imputation so that these data points can still be used to train and validate the classifier. If the feature with the most missing values - `['Cabin']` is dropped altogether before checking for NaNs, we can retain 712 samples at the cost of a reduced feature space. One approach is to impute all missing data to "guess" what the missing data might be. An alternative is to set a threshold and only remove columns and rows with a higher percentage of missing values than the threshold, then perform some form of data imputation on the remaining NaNs [2].

Figure 1: Pairplot showing pairwise relationships between features

UMAP (Uniform Manifold Approximation and Projection) reduction was attempted on scaled data, but it didn't help much since this dataset is not of an extremely high dimension [3].

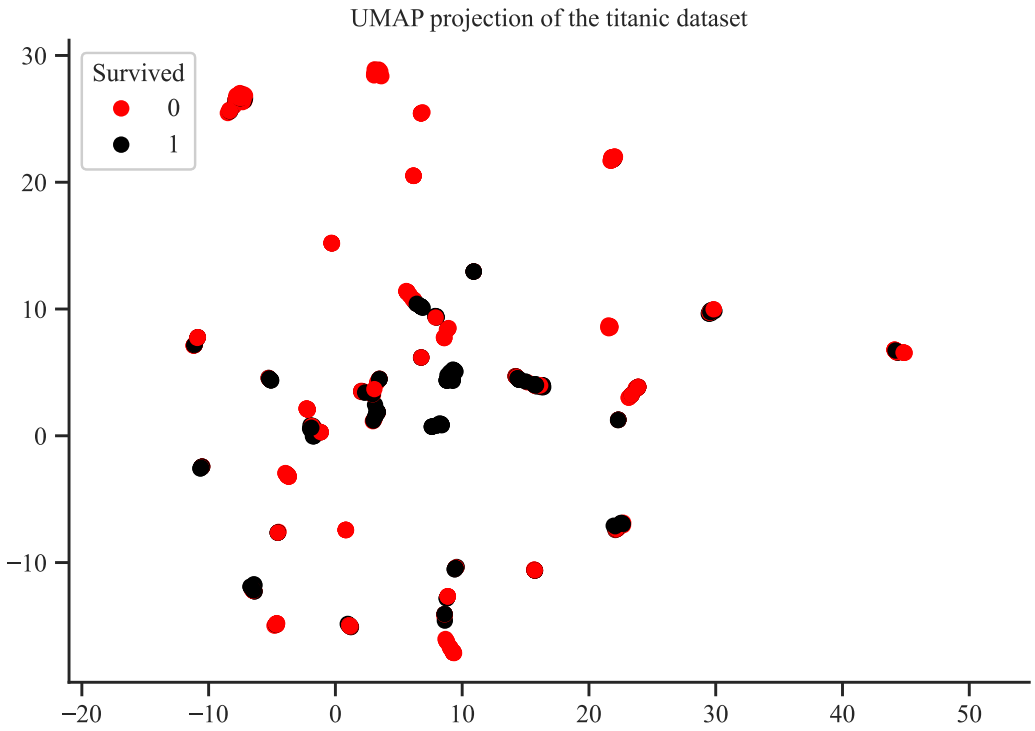

Figure 2: UMAP scatterplot showing reduced dimensionality

It isn't clear, based on the UMAP reduction, whether or not a linear relationship can be derived from the data given and there isn't much separability between the reduced dimension data points.

3 Challenges

The main challenges faced with this dataset were concerning what to do about missing values and picking out which features were relevant to the classification. I have a feeling that `['Cabin']` is a relevant feature, but when using a missing value threshold of 0.7 it was dropped from the model. I struggled with this aspect in the model design and this is definitely something to improve upon. Another design parameter which could be tweaked are the bins to use for `['Age Group']` and `['Fare Group']`, but I am not sure how to reason through this step theoretically.

4 Approach

4.1 Main workflow

1. Load data
2. Preprocessing
3. Separate data into training and validation sets
4. Use training set to fit the Ridge Classifier model with K=2 (binary classifier)
5. Use the fitted model to make predictions on the validation set

6. Use the validated model to make predictions on the testing set
7. Evaluate the model's performance

4.2 Preprocessing steps

1. Check to see if any columns contain missing values.
2. Set a threshold for an acceptable amount of missing values per column and row.
3. Remove columns (features) and rows (points) with more than the threshold of missing values.
4. Fill remaining missing values with zeros. Other imputation techniques are also possible.
5. Determine suitable bin sizes for continuous features such as `['Age']` and `['Fare']` and bin them accordingly.
6. One-hot encode categorical features.
7. Standardize numerical features (if necessary).

4.3 Data encoding

Age was broken into six categories:

Age	Age Group	Count	Percentage
(0, 2]	Baby	5	2.73%
(2, 12]	Child	6	3.28%
(12, 18]	Adolescent	12	6.56%
(18, 30]	Young Adult	46	25.14%
(30, 65]	Adult	104	56.83%
(65, 99]	Elderly	10	5.46%

Table 1: Bins for Age Group feature transformation

Fare amount was similarly broken into six categories:

Fare	Fare Group	Count	Percentage
(0, 25]	Cheapest	21	11.60%
(25, 50]	Cheaper	48	26.52%
(50, 100]	Cheap	71	39.23%
(100, 200]	Pricy	25	13.81%
(200, 500]	Pricier	14	7.73%
(500, 600]	Priciest	2	1.10%

Table 2: Bins for Fare Group feature transformation

Then the data was split into training and validation sets in order to benchmark the model. The test subset was set to be 1/3 of the total samples in `train.csv`.

There was a tradeoff between keeping all features and eliminating missing data. The data seems to suggest that the model would benefit from mapping the data using a nonlinear function in order to create separability between classes.

5 Evaluation and summary

In order to benchmark the model, it is assumed that it should have better performance than just predicting survival based on `['Sex'] == Female`. Table 1 and 2 show the error for the baseline model and the ridge regression model when rows and columns with more than 70% NaNs are dropped, and remaining missing values are filled with zeros.

Training error	Validation error	Kaggle error
21.81%	20.34%	23.44%

Table 3: Baseline training, validation, and Kaggle (test) error (females survive)

Training error	Validation error	Kaggle error
17.79%	18.31%	23.68%

Table 4: Training, validation, and Kaggle (test) error for ridge regression model

	Predicted	
Actual	0	1
0	152	23
1	31	89

Table 5: Confusion matrix for validation set

Metric	Score
Accuracy	0.82
Precision	0.79
Recall	0.74

Table 6: Metrics for validation set

The ridge regression model performed slightly better on the training and validation sets but suffered on the test set.

It can be observed from Table 3 that the model did slightly worse at predicting survival than it did at predicting death, although this can be partially explained by the fact that the data is biased toward death (in the train set, 549 out of 891 passengers died versus 342 that lived). Nonlinear models would likely perform better, unless the features can be properly engineered. I was not able to achieve much better performance than the gender-based model which assumes that all females survived. The ridge regression classifier might benefit from added bias, which could be implemented fairly easily.

6 What I learned

A pairplot was used to visualize the possible relationships between pairwise features [3]. UMAP was also used to try to visualize the data in a reduced dimension, but it didn't produce any insightful results due to the relatively low dimensionality of the data [3] Data imputation techniques are something to consider that may marginally improve performance [2]. I think feature engineering is a huge part of this data set, and likely for most datasets taken from the real world. I don't have the metrics to back it up, but I observed that a small increase in performance was achieved by transforming the continuous features such as `['Age']` and `['Fare']` into discrete, categorical features [8]. This is something I didn't anticipate, but made sense when considering that discrete binned categories are much easier to analyze for patterns.

References

[1] (2018) Titanic: Machine Learning from Disaster. [Online]. Available: https://www.kaggle.com/c/titanic/overview

[2] (2019) Fundamental Techniques of Feature Engineering for Machine Learning. [Online]. Available: https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114#3abe

[3] (2018) Document embedding using UMAP. [Online]. Available: https://umap-learn.readthedocs.io/en/latest/document_embedding.html

[4] (2016) A comprehensive introduction to data wrangling. [Online]. Available: https://www.springboard.com/blog/data-wrangling/

[5] (2019) Ml explainability: Deep dive into ml model! [Online]. Available: https://www.kaggle.com/niyamatalmass/ml-explainability-deep-dive-into-the-ml-model

[6] (2021) Titanic Tutorial. [Online]. Available: https://www.kaggle.com/code/alexisbcook/titanic-tutorial/notebook

[7] (2017) Data Wrangling with Python Tutorial. [Online]. Available: https://github.com/Rogerh91/Springboard-Blog-Tutorials/blob/master/Data%20Wrangling%20with%20Python%20Tutorial/Data%20Wrangling%20with%20Python%20Tutorial%20(Springboard).ipynb

[8] (2019) Pandas Cut - Continuous to Categorical. [Online]. Available: https://www.absentdata.com/pandas/pandas-cut-continuous-to-categorical/

Code: mp01.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 19 12:55:30 2022

@author: teen
"""

import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import bokeh
import umap
import os
from sklearn.linear_model import LogisticRegression, LinearRegression, Ridge, RidgeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics

def DrawArrayAxes(nRows, nColumns, iRow, iColumn,
                  axisBoundaries=[0.02,0.02,0.13,0.10],
                  horizontalBuffer=0.02,
                  verticalBuffer=0.09,
                  sharex=None,
                  figure=None):
    if not figure:
        figure = plt.gcf()
    topArrayEdge = axisBoundaries[0]
    rightArrayEdge = axisBoundaries[1]
    bottomArrayEdge = axisBoundaries[2]
    leftArrayEdge = axisBoundaries[3]
    axesWidth = (1-leftArrayEdge-rightArrayEdge
                      -(nColumns-1)
                      *horizontalBuffer)/nColumns
    axesHeight = (1-topArrayEdge-bottomArrayEdge-(nRows-1)*verticalBuffer)/nRows
    leftEdge = leftArrayEdge  +(iColumn -1)*(axesWidth+horizontalBuffer)
    bottomEdge = bottomArrayEdge+(nRows-iRow)*(axesHeight+verticalBuffer)

    axes = figure.add_axes([leftEdge,
                            bottomEdge,
                            axesWidth,
                            axesHeight], sharex=sharex)
    axes.spines['top'].set_visible(False)
    axes.spines['right'].set_visible(False)
    axes.yaxis.set_ticks_position('left')
    axes.xaxis.set_ticks_position('bottom')
```

```
    return (axes)

def FormatFigureText(figure=None, fontSize=None, fontName=None):
    if not figure:
        figure = plt.gcf()
    textHandles = [h for h in figure.findobj() if type(h) == matplotlib.text.Text]
    for th in textHandles:
        if fontName:
            th.set_fontname(fontName)
        if fontSize:
            th.set_fontsize(fontSize)

def encode_one_hot(y):
    '''
    Parameters
    ----------
    y : vector of integers
        categorically integer-encoded vector to encode into one-hot encoding.

    Returns
    -------
    n_classes : integer
        number of classes for one-hot encoding.
    y_encoded : y x n_classes sized-array
        one-hot encoding of y.

    '''
    n_classes = len(np.unique(y))
    y_encoded = np.eye(n_classes)[y]

    return n_classes, y_encoded

def ridge_classification(X, y, reg_param, train=False, W=None):
    '''
    Parameters
    ----------
    X : array
        data with which to train or test classifier.
    y : array
        labels with which to train or test classifier.
    reg_param : float
        regularization parameter, sometimes called lambda or alpha.
    train : bool
        whether classifier should train or test
    W : array
        estimate for W, None if classifier is training. Must be supplied if testing

    Returns
    -------
    W :
```

```
        array of estimate for W
    y_pred : array
        array of labels returned by classifier.

    '''
    if train:
        n_features = np.size(X, 0)
        W = np.linalg.inv(X @ X.T + reg_param * np.eye(n_features)) @ X @ y

    ys = X.T @ W
    y_pred = np.argmax(ys, 1)

    return W, y_pred

def classification_error(y, y_pred):
    '''
    Parameters
    ----------
    y : array of ints
        true labels.
    y_pred : array of ints
        labels returned by classifier.

    Returns
    -------
    error : float
        classifier error.

    '''
    n_samples = np.size(y)
    errors = y_pred-y
    n_errors = np.count_nonzero(errors)
    error = 1 - (n_samples - n_errors) / n_samples

    return error

sns.set(style='white', context='notebook', rc={'figure.figsize':(14,10)})
#%% Load data

cwd = '/Users/teen/School/EE 510/hw/mp01/'
data_path = 'data/titanic/'
figure_path = "out/"
train_all_df = pd.read_csv(cwd + data_path + "train.csv")
test_all_df = pd.read_csv(cwd + data_path + "test.csv")

#%% Preprocessing
# Determine if NaNs exist in training dataset
pd.isnull(train_all_df).any()

threshold = 0.7
# #Dropping columns with missing value rate higher than threshold
```

```
train_df = train_all_df[train_all_df.columns[train_all_df.isnull().mean() < threshold]]
test_df = test_all_df[test_all_df.columns[test_all_df.isnull().mean() < threshold]]


# #Dropping rows with missing value rate higher than threshold
train_df = train_df.loc[train_df.isnull().mean(axis=1) < threshold]
test_df = test_df.loc[test_df.isnull().mean(axis=1) < threshold]

pairplot = sns.pairplot(train_df, hue="Survived")

FormatFigureText(fontSize=11, fontName='Times New Roman')
saveName = cwd + figure_path + "pairplot.pdf"
plt.savefig(saveName, bbox_inches="tight")

# Only use the cabin letter
# cabin_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
# for i, cabin_letter in enumerate(cabin_list):
#     mask = train_df['Cabin'].str.startswith(cabin_letter)
#     train_df.loc[mask, 'Cabin'] = cabin_letter

# Features to select, get dummies
features = ['Pclass','Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']

X = train_df[features]
X_test = test_df[features]

X = X.fillna(0)
X_test = X_test.fillna(0)

# Encoding age
# 0 : baby 0-2 yrs
# 1 : child 2-12 yrs
# 2 : adolescent 12-18
# 3 : adult 18-65
# 4 : elderly 65-

category = pd.cut(X.Age, bins=[0,2,12,18,30,60,99], labels=['Baby',
                                                            'Child',
                                                            'Adolescent',
                                                            'Young Adult',
                                                            'Adult',
                                                            'Elderly'])
X.insert(3, 'Age Group', category)
X = X.drop(columns='Age')

category = pd.cut(X.Fare, bins=[0,25,50,100,200,500,600],labels=['Cheapest',
                                                                 'Cheaper',
                                                                 'Cheap',
                                                                 'Pricy',
                                                                 'Pricier',
                                                                 'Priciest'])
```

```
X.insert(7, 'Fare Group', category)
X = X.drop(columns='Fare')

categorical_columns = ['Sex','Pclass','Age Group', 'Fare Group', 'Embarked']
for column in categorical_columns:
    encoded_columns = pd.get_dummies(X[column], prefix=column)

    X = pd.merge(
        left=X,
        right=encoded_columns,
        left_index=True,
        right_index=True,
    )

    X = X.drop(columns=column)

category = pd.cut(X_test.Age, bins=[0,2,12,18,30,60,99], labels=['Baby',
                                                                 'Child',
                                                                 'Adolescent',
                                                                 'Young Adult',
                                                                 'Adult',
                                                                 'Elderly'])
X_test.insert(3, 'Age Group', category)
X_test = X_test.drop(columns='Age')

category = pd.cut(X_test.Fare, bins=[0,25,50,100,200,500,600],labels=['Cheapest',
                                                                      'Cheaper',
                                                                      'Cheap',
                                                                      'Pricy',
                                                                      'Pricier',
                                                                      'Priciest'])
X_test.insert(7, 'Fare Group', category)
X_test = X_test.drop(columns='Fare')

categorical_columns = ['Sex','Pclass','Age Group', 'Fare Group', 'Embarked']
for column in categorical_columns:
    encoded_columns = pd.get_dummies(X_test[column], prefix=column)

    X_test = pd.merge(
        left=X_test,
        right=encoded_columns,
        left_index=True,
        right_index=True,
    )

    X_test = X_test.drop(columns=column)

X = X.drop(columns='Embarked_0')
# tempdf = pd.get_dummies(X[column], prefix=column)

y = train_df["Survived"]
```

```
# %% Split train data into train and validation sets
X_train, X_validation, y_train, y_validation = train_test_split(X,
                                                                y,
                                                                test_size=0.33,
                                                                random_state=42)


X_data = X.values
# X_validation_data = X_validation.values

scaled_X_data = StandardScaler().fit_transform(X_data)

reducer = umap.UMAP(random_state=42)

reducer.fit(scaled_X_data)
embedding = reducer.embedding_
embedding.shape

fig = plt.figure()
fig.set_size_inches(6.5, 4.5)
fig.clf()

ax = DrawArrayAxes(1, 1, 1, 1)

umap_scatter = ax.scatter(
    embedding[:, 0],
    embedding[:, 1],
    c=y,
    cmap=matplotlib.colors.ListedColormap(['red', 'black']))
plt.gca().set_aspect('equal', 'datalim')
plt.title('UMAP projection of the titanic dataset', fontsize=24)

legend = ax.legend(*umap_scatter.legend_elements(),
                   loc="upper left", title="Survived")

ax.add_artist(legend)

FormatFigureText(fontSize=11, fontName='Times New Roman')
saveName = cwd + figure_path + "UMAP_scatter.pdf"
plt.savefig(saveName, bbox_inches="tight")

# %% Create baseline by using Sex to predict survival, 1 if Sex_Female = True

base_train_error = classification_error(y_train, X_train.Sex_female)
base_test_error = classification_error(y_validation, X_validation.Sex_female)

print("\nBaseline training error (females survive): %2.2f%%" % (100.0 * base_train_error))
print("Baseline validation error (females survive): %2.2f%%" % (100.0 * base_test_error))


# %% Fit the model with Linear Regression
```

```
model = LinearRegression()
model.fit(X_train, y_train)

# predictions = model.predict(X)

train_score = model.score(X_train, y_train)
test_score = model.score(X_validation, y_validation)

train_error = 1-train_score
test_error = 1-test_score

# %% Fit the model with Logistic Regression

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# predictions = model.predict(X)

train_score = model.score(X_train, y_train)
test_score = model.score(X_validation, y_validation)

train_error = 1-train_score
test_error = 1-test_score


# %% Fit the model with Ridge Regression

model = RidgeClassifier(alpha=1e-4,tol=1e-2,max_iter=1000,fit_intercept=True)
model.fit(X_train, y_train)

y_predicted = model.predict(X_validation)

y_actual = pd.Series(y_validation, name='Actual')
y_predicted = pd.Series(y_predicted, name='Predicted')

c_matrix = metrics.confusion_matrix(y_actual, y_predicted)
print("Confusion matrix\n")
print(c_matrix)
print("Accuracy score: %f" % metrics.accuracy_score(y_actual, y_predicted))
print("Precision score: %f" % metrics.precision_score(y_actual, y_predicted))
print("Recall score: %f" % metrics.recall_score(y_actual, y_predicted))


train_score = model.score(X_train, y_train)
test_score = model.score(X_validation, y_validation)

train_error = 1-train_score
test_error = 1-test_score

print("\nRidge regression training error: %2.2f%%" % (100.0 * train_error))
```

```
print("Ridge regression validation error: %2.2f%%" % (100.0 * test_error))

# output = pd.DataFrame({'PassengerId': test_df.PassengerId, 'Survived': predictions})
# output.to_csv(cwd + 'out/' + 'submission.csv', index=False)
# print("Your submission was successfully saved!")
```