Wisconsin Breast Cancer Diagnosis Deep Learning Revisited

By

Yuefeng Zhang

November 2017

1. Definition

1.1 Project Overview

The machine learning methodology has long been used in medical diagnosis [1]. The Wisconsin Breast Cancer Database (WBCD) dataset [2] has been widely used in research experiments.

Most of publications focused on traditional machine learning methods such as decision trees and decision tree-based ensemble methods [5].

Recently supervised deep learning method starts to get attention. For instance, Stahl [3] and Geekette [4] applied this method to the WBCD dataset [2] for breast cancer diagnosis using feature values calculated from digitized image of a Fine Needle Aspirate (FNA) of a breast mass. These features describe the characteristics of the cell nuclei present in the image.

Stahl [3] used the WBCD dataset with derived features (e.g., mean, standard error, …, etc.) and experimented three types of deep neuron networks: 1, 2, and 3 hidden layers of 30 neurons without any data pre-processing. Geekette [4] used only the originally identified features with pre-processing such as center and scale, but did not provide detailed information of the data preprocessing or the neuron network architecture in use.

This capstone project report presents a new supervised deep learning method for analyzing the same WBCD dataset [2] for breast cancer diagnosis using common open source libraries. The new supervised deep learning method inherits the merits of the methods experimented by Stahl [3] and Geekette [4]. Specifically, similarly to [4], the new method uses the originally identified features [2], pre-processes data using center and scale, and treats the breast cancer diagnosis problem as a 2-class classification problem. Like [3], the new method adopts multiple (three in this capstone project) hidden layers of deep neuron network.

1.2 Problem Statement

Given a list of features (i.e., feature vectors) calculated from a digitized image of the FNA of a breast mass from a patient (see Section 2.1 for details), the problem is how to diagnose (determine) whether or not the patient has breast cancer.

Similarly to the practices by Stahl [3] and Geekette [4], this problem is treated as a 2-class (B – *benign,* M - *malignant*) classification problem in this capstone project.

The strategy for solving this classification problem is to utilize a new deep learning network with multiple hidden and dropout layers (See Section 2.3 for details). The inputs to the new deep learning network are the feature values computed from digitized image of FNA, and the outputs from the network are the two identified classification diagnosis classes B or M.

1.3 Metrics

Similarly to the evaluation method practiced by Geekette [4], The prediction accuracy is used as the major evaluation metrics in this capstone project report.

Specifically, the Keras binary accuracy metrics API (https://keras.io/metrics/) is used to calculate the mean accuracy rate across all predictions for the WBCD diagnosis binary classification problem as follows:

$$\textit{mean accuracy rate} = (m / n) * 100\%$$

where $\boldsymbol{n}$ is the total number of testing samples and $\boldsymbol{m}$ is the total number of predictions that match the labeled responses for testing.

2. Analysis

2.1 Data Exploration

The dataset used in this project is the publically available WBCD dataset [2]: https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data

As described in the WBCD dataset [1], the following features were computed for each cell nucleus and will be used as inputs to the new deep learning network:

- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- Marginal Adhesion
- Single Epithelial Cell Size
- Bare Nuclei
- Bland Chromatin
- Normal Nucleoli
- Mitoses

The following table is a statistical summary of the dataset.

	CT	UCSize	UCShape	MA	SECSize	BN	BC	NN	Mitoses	Diagnosis
count	699.000000	699.000000	699.000000	699.000000	699.000000	683.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.544656	3.437768	2.866953	1.589413	2.689557
std	2.815741	3.051459	2.971913	2.855379	2.214300	3.643857	2.438364	3.053634	1.715078	0.951273
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000	2.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000	2.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.000000	4.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.000000

CT	- Clump Thickness
UCSize	- Uniformity of Cell Size
UCShape	- Uniformity of Cell Shape
MA	- Marginal Adhesion
SECSize	- Single Epithelial Cell Size
BN	- Bare Nuclei
BC	- Bland Chromatin
NN	- Normal Nucleoli

Table 1: Statistical summary of the original WBCD dataset.

The following issues with the dataset are observed:

- Missing data
- Small dataset size
- Various ranges of data values
- Unbalanced data
- Skewed data

One of the common issues with dataset is missing data and there is no exception to the WBCD dataset. Specifically, as shown in Table 1, there are 16 Bare Nuclei entries missing (marked as ? in the original WBCD dataset).

The WBCD dataset [2] consists of only 699 samples, which is too small for deep learning after dividing the dataset into training and testing subsets.

The ranges of feature values for different features are different. The average feature values change from about 1.5 to 4.5 (see Section 2.2 for the visualization of value ranges).

The total numbers of data samples with different labels are not balanced. In fact, the total number of data samples labeled as B is almost doubled the total number of data samples labeled as M (see Section 2.2 for the visualization of those numbers).

Another common issue associated with the dataset is skewness (see Section 2.2 for details).

The visualizations of those dataset issues are provided in Section 2.2 and the solutions to these dataset issues are described in Section 3.1.

2.2 Exploratory Visualization

This section shows some of the dataset issues described in Section 2.1 via data visualization.

Figure 1 shows the various mean value ranges of different features as described in Section 2.1.

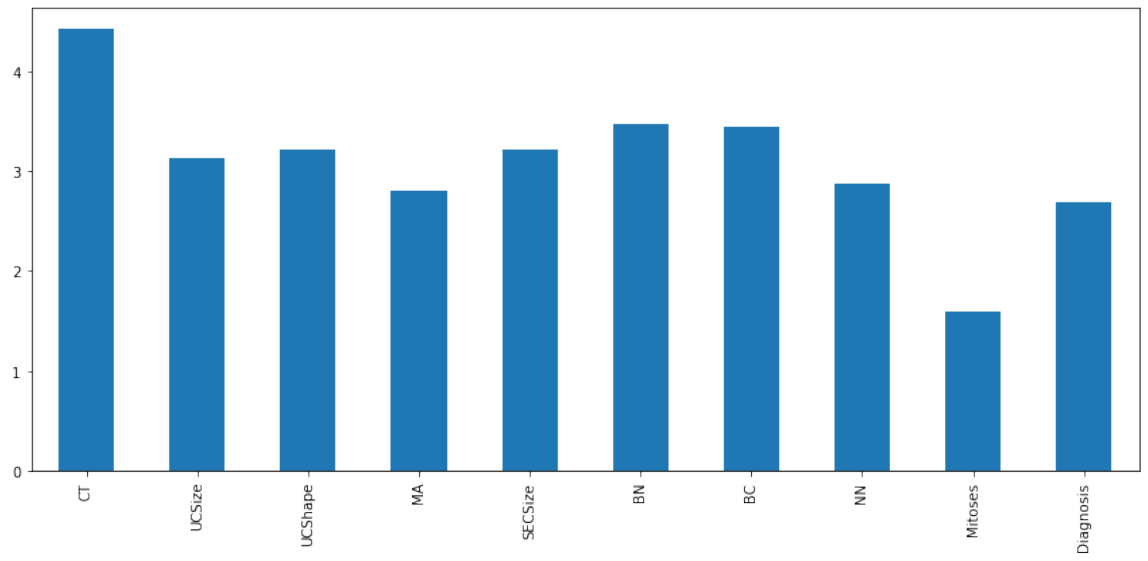

Figure 1: Various ranges of feature values.

To visualize the unbalanced dataset issue, Figure 2 shows the total numbers of samples labeled as B and M.

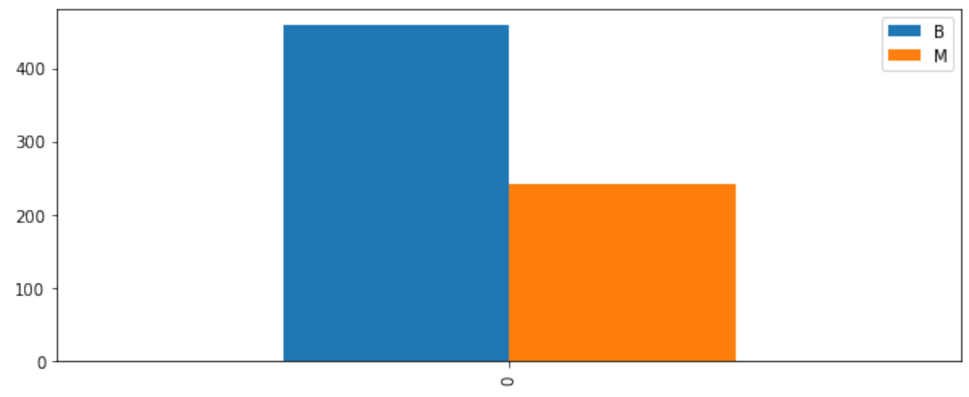

Figure 2: The total numbers of data samples labeled as B (blue) and M (orange) in the original dataset.

In order to visualize the data skewness issue, Figure 3 shows the Pandas generated scatter plot matrix of the dataset after data pre-processing (see Section 3.1 for details). It can be seen from the figure that the dataset is significantly skewed to the right.

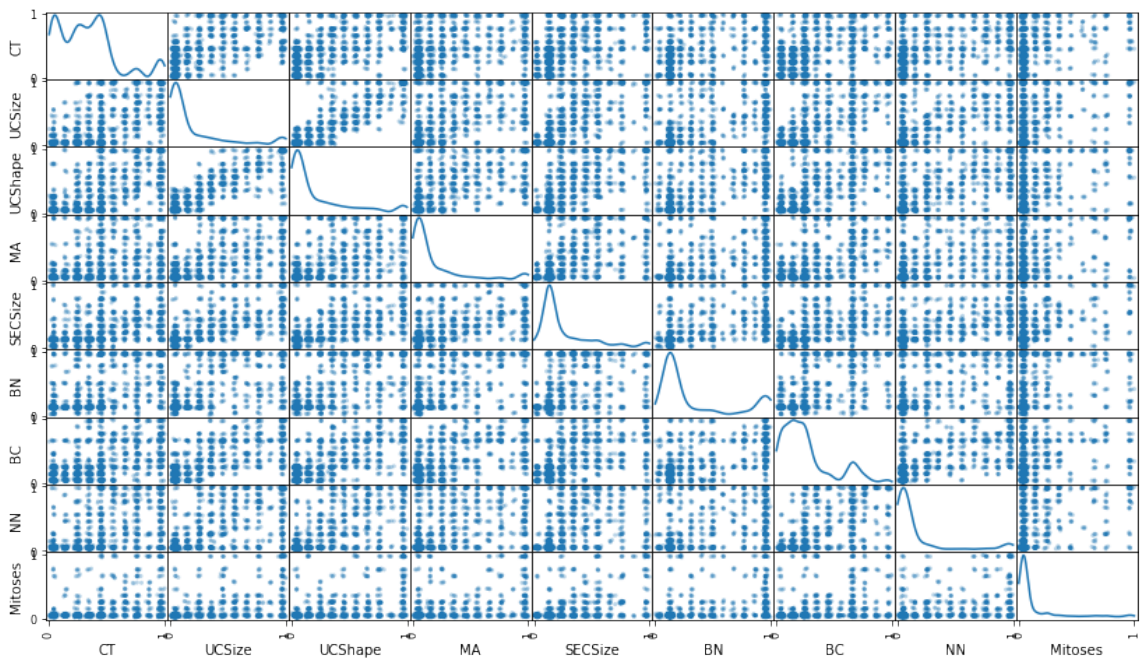

Figure 3: Skewness of the dataset.

2.3 Algorithms and Techniques

As described before, the breast cancer diagnosis problem is treated as a 2-class (*benign* or *malignant*) classification problem in this capstone project.

A new supervised deep learning network is used for the classification. The architecture of the new deep learning network is shown in Figure 4.

This new supervised deep learning network inherits the merits of the methods experimented by Stahl [3] and Geekette [4].

Specifically, similarly to [4], the new method uses the originally identified features [2], pre-processes data using center and scale (see Section 3.1 for details), and treats the breast cancer diagnosis problem as a 2-class (*benign* or *malignant*) classification problem.

Like [3], the new method adopts multiple (three in this capstone project) hidden layers of deep neuron network.

In addition, the dataset unbalance and skewness problems are handled as part of data preprocessing as well.

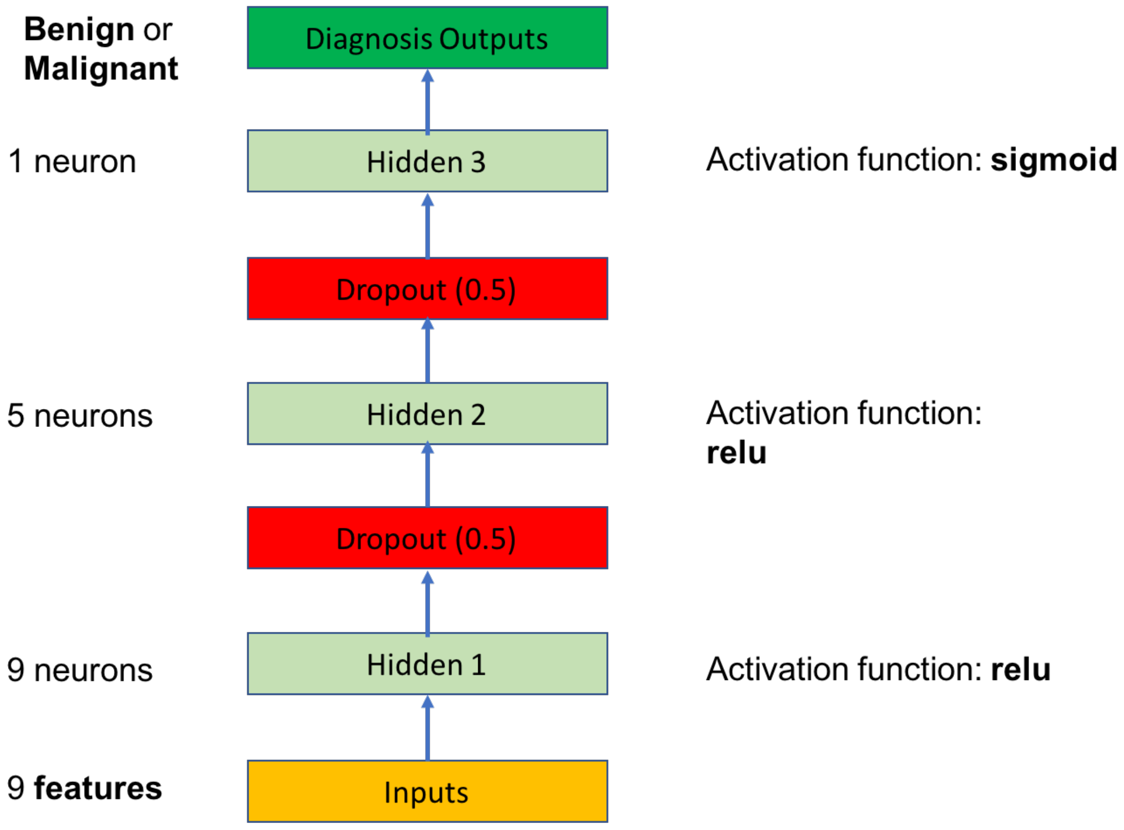

Figure 4: The architecture of the new deep neuron network.

As shown in Figure 4, the inputs to the new deep learning network are the feature values computed from digitized image of FNA, and the outputs from the network are the two identified classification diagnosis classes B (*benign*) or M (*malignant*).

The first hidden layer uses 9 neurons and the last hidden layer uses 1 neuron to match the number of input features and the number of output classes respectively. The size (5 neurons) of the hidden layer in the middle is in the middle of the sizes of the first and last hidden layers.

In addition, dropout layers with dropout rate of 50% are introduced between hidden layers to avoid overfitting and potentially improve the new deep learning network performance in accuracy.

The common rectified linear unit (ReLU) activation function *relu* is used in hidden layers 1 and 2. The widely used *sigmoid* activation function is used in the final hidden layer 3 to produce continuous output in the range of (0, 1). A threshold of 0.5 is inherently used to generate binary diagnosis output from the continuous output.

The following loss function, optimizer, and metrics functions are used in training the new deep learning network model:

- **Loss function:** binary cross entropy
- **Optimizer function:** Adam (Adaptive Moment Estimation)
- **Metrics function:** accuracy

The binary cross entropy loss function is chosen because the WBCD classification is a binary classification problem.

The accuracy rather than binary accuracy metrics function is chosen because the accuracy metrics function supports accuracy history in model training, but the binary accuracy metrics function does not.

The Adam optimizer function is chosen because it is a replacement optimization algorithm for stochastic gradient descent (SGD) for training deep learning models.

More information of the new deep learning network architecture is provided in Section 3.2.

2.4 Benchmark

The result of the new supervised deep learning network will be compared with the result of applying the open source scikit-learn out-of-box implementation of the Random Forest Classifier [6] with default settings to measure the relative performance.

3. Methodology

3.1 Data Preprocessing

The missing data entries are simply replaced with 0 in this capstone project for the following reason. The number (16 in total) of missing entries is relatively small and thus replacing them with 0 should not introduce noticeable impact to statistical data patterns of the dataset.

As described in Section 2.1, the total numbers of data samples with different labels are not balanced. This unbalanced data issue is resolved by generating new data samples labeled as M. Specifically, the set of data samples labeled as M is doubled by introducing normal distributed random noise to the existing data samples with label M. As shown in Figure 5, the resulting new set of data samples with label M is reasonably balanced with the set of data samples labeled as B.

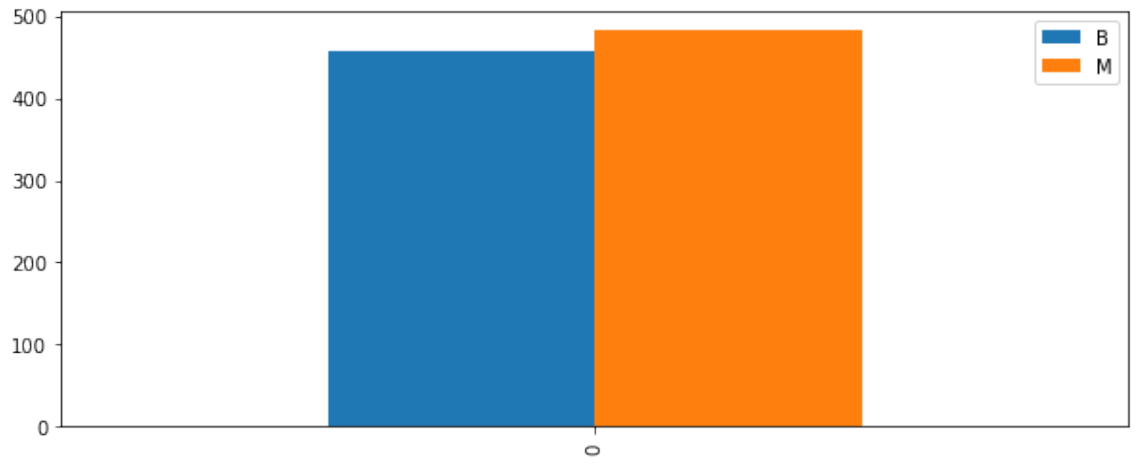

Figure 5: The total numbers of data samples labeled as B (blue) and M (orange) after adding new data samples with labels M.

As described Section 2.1, the original dataset consists of only 699 samples. This small data size issue is resolved in this capstone project by generating new data samples as follows.

First the data samples labeled as B are separated from the data samples labeled as M. Then the set of data samples labeled as M is doubled by introducing normal distributed random noise. After that, the set of data samples labeled as B are combined with the new expanded set of data samples labeled as M. Finally, similarly to expanding the set of data samples labeled as M, the new combined dataset is doubled by introducing normal distributed random noise.

The noise generation is achieved by using the Numpy random number generation library *Numpy.random.normal*(*mean, sigma, features.shap*e) to generate an array of normal distributed random numbers with the same feature dimensions as a given dataset, where *mean* = 0 and the standard deviation *sigma* = 0.1. After that, the generated array of random numbers are added into the given dataset element by element to form a new dataset.

The values in the last column "Diagnosis/Class" in the WBCD dataset [2] are used as labels in training the new deep learning model in this capstone project. These labels are reused to label the corresponding samples (feature vectors) in the new generated dataset by combining the new generated array of feature values with the "Diagnosis/Class" column from the given dataset into a new dataset.

After dataset expansion, the final new dataset contains 1,880 samples in total for model training and testing in this capstone project.

As a common practice, for each of the features, the feature values are scaled into the range of [0, 1] for deep learning as follows:

$$(Value - Minimum) / (Maximum - Minimum)$$

The identified data skewness issue is handled by taking squared root of the feature values. This is achieved via Numpy API as follows:

$$Value = Numpy.sqrt(Value)$$

Figure 6 shows the result of applying the squared root transformation to the dataset.

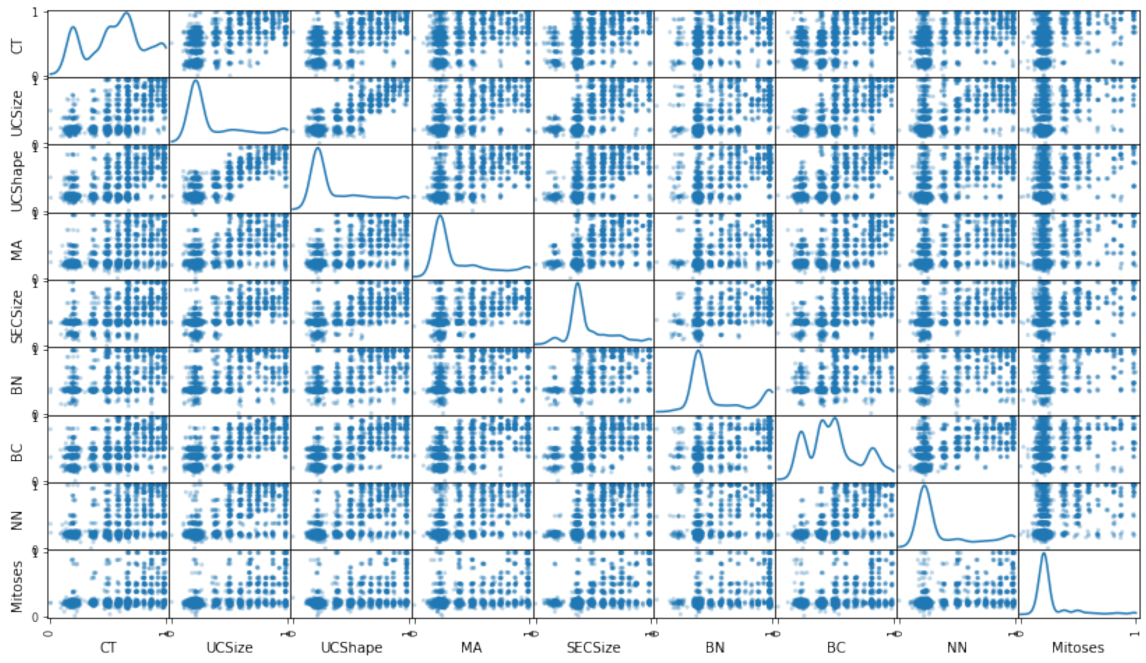

Figure 6: Transformed dataset.

3.2 Implementation

The following common open source libraries are used to implement the new deep learning network:

- Numpy

- Pandas
- scikit-lean
- Keras
- matplotlib

The rest of this section describes the specific implementation details related to this capstone project.

3.2.1 Data Loading

As the first step of the implementation, the dataset csv file is loaded as follows using Pandas API:

```
import pandas as pd
headers = ["ID","CT","UCSize","UCShape","MA","SECSize","BN","BC","NN","Mitoses","Diagnosis"]
data = pd.read_csv('breast-cancer-wisconsin.csv', na_values='?', header=None, index_col=['ID'], names = headers)
data = data.reset_index(drop=True)
data = data.fillna(0)
```

The missing data issue is handled in data loading in two steps. First the missing data question mark "?" is replaced with NaN. Then those NaN values are filled with 0.

3.2.2 Dataset Expansion

In order to resolve the data unbalance and small dataset size issues, given a dataset, the following function is defined to generate a new dataset of the same dimensions by introducing normal distributed random noise.

```
def generateData(data1):
    # make a copy of data
    data2 = data1.copy()

    # separate labels from features
    diagnosis2 = data2['Diagnosis'] # labels
    features2 = data2.drop(['Diagnosis'], axis = 1)
    features2_headers = ["CT","UCSize","UCShape","MA","SECSize","BN","BC","NN","Mitoses"]

    mean, sigma = 0, 0.1
    # creating a noise with the same dimension as the dataset
    noise = np.random.normal(mean, sigma, features2.shape)
    features2 = features2.apply(pd.to_numeric, errors='ignore')
    features2_with_noise = features2.add(pd.DataFrame(noise, columns = features2_headers), fill_value=0)
    data2 = pd.concat([features2_with_noise, pd.DataFrame(diagnosis2)], axis = 1)

    return data2    return data2
```

The above dataset generation function is called as following to combine a new generated dataset with the given dataset.

```
new_data = generateData(data)
data = data.append(new_data, ignore_index=True)
```

3.2.3 Scaling Dataset

As described before, the values of different features have different ranges and majority of those values are outside the range of [0, 1]. However, the new deep learning network requires the input values in the range of [0, 1]. To this end, the values of the dataset are scaled into the range of [0, 1] as follows:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
numerical = ["Diagnosis","CT","UCSize","UCShape","MA","SECSize","BN","BC","NN","Mitoses"]
data[numerical] = scaler.fit_transform(data[numerical])
```

3.2.4 Handling Dataset Skewness

As described before, the dataset skewness issue is handled by separating features from labels and then applying squared root transformation to the feature values only. This is achieved as follows:

```
diagnosis = data['Diagnosis']
features = data.drop(['Diagnosis'], axis = 1)
sqrt_features = features.copy()
for feature_name in sqrt_features.columns:
    sqrt_features[feature_name] = np.sqrt(sqrt_features[feature_name])
# convert numpy ndarray into Pandas DataFrame
features = pd.DataFrame(sqrt_features)
```

3.2.5 Splitting Dataset into Training and Testing Subsets

As described before, the dataset is split into two parts: 75% for model training and 25% for model testing:

```
from sklearn.model_selection import train_test_split
# Shuffle and split the data into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(features, diagnosis, test_size=0.25, random_state=42)
```

3.2.6 Reformatting Dataset

After splitting dataset, the resulting data subsets must be re-indexed as follows to avoid dataset key mismatch issue:

```
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

Then the resulting Pandas DataFrames must be converted into Numpy arrays as below because the Pandas DataFrame is not supported by Keras API.

```
X_train = X_train.values
y_train = y_train.values
X_test  = X_test.values
y_test  = y_test.values
```

3.2.7 Benchmark

The scikit-learn out-of-box implementation of the Random Forest Classifier is used to produce the benchmark prediction result for comparison as follows:

```
from sklearn.ensemble import RandomForestClassifier
# Create model
rfc = RandomForestClassifier()
# Train model
rfc.fit(X_train, y_train)
# Test model
score = rfc.score(X_test, y_test)
```

The following are the default settings of the Random Forest Classifier:

- *n_estimators=10*,
- *criterion='gini'*,
- *max_depth=None*,
- *min_samples_split=2*,
- *min_samples_leaf=1*,
- *min_weight_fraction_leaf=0.0*,
- *max_features='auto'*,
- *max_leaf_nodes=None*,
- *min_impurity_decrease=0.0*,
- *min_impurity_split=None*,
- *bootstrap=True*,
- *oob_score=False*,
- *n_jobs=1*,
- *random_state=None*,
- *verbose=0*,
- *warm_start=False*,
- *class_weight=None*

3.2.8 Implementation of the New Deep Learning Network

The popular open source Keras deep learning library [7] is utilized for the implementation of the new deep learning network:

```
# Import libraries
from keras.layers import Dense
from keras.layers import Dropout
from keras.models import Sequential
import keras
import keras.utils
from keras import utils as np_utils

# Define DNN architecture
model = Sequential()
model.add(Dense(9, activation='relu', input_dim=9))
model.add(Dropout(0.5))
model.add(Dense(5, activation='relu', input_shape=(9,)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid', input_shape=(5,)))
# Compile DNN model
model.compile(loss='binary_crossentropy',
        optimizer='Adam',
        metrics=['accuracy'])
# Train DNN
model.fit(X_train, y_train, epochs=500, batch_size=32)
# Test DNN
```

```
score = model.evaluate(X_test, y_test, batch_size=32)
```

The Jupyter notebook of the new deep learning network is available in GitHub [8].

3.3 Refinement

3.3.1 Depth of Network

This capstone project used a relatively small dataset of 1,880 samples. In this case, using the minimum three hidden layers of deep learning network is appropriate to get the best performance.

As shown in the table below, using more than three hidden layers negatively impacts the performance in accuracy because there is no enough data to train the deep learning network.

Hidden Layers	Dropouts	Epochs	Batch Size	Accuracy
9, 5, 1	**0.5**	**500**	**16**	**0.974468**
9, 9, 5, 1	0.5	500	16	0.965714

3.3.2 Dropout

In theory, adding Dropout layers should help to avoid overfitting and thus potentially improve deep learning network performance in accuracy. However, according to the experimental results, no noticeable performance gain was achieved by using Dropouts of rate 50% once the number of epochs is 500 or more.

As shown in the table below, when Dropouts are used, the dropout rate of 50% seems to be a sweet spot. For example, with all other configuration parameters fixed, using dropout of 30% produced a prediction accuracy of about 97.1428%, while using dropout rate of 50% obtained 97.7143% prediction performance in accuracy.

Hidden Layers	Dropouts	Epochs	Batch Size	Accuracy
9, 5, 1	**0.5**	**500**	**16**	**0.974468**
9, 5, 1	0.3	500	16	0.971428

3.3.3 Batch Size

The accuracy performance of the new deep learning network is sensitive to the batch size. As shown in the table below, a batch size of 16 seems to work the best.

Hidden Layers	Dropouts	Batch Size	Epochs	Accuracy
9, 5, 1	0.5	10	500	0.948571
9, 5, 1	**0.5**	**16**	**500**	**0.9744468**
9, 5, 1	0.5	32	500	0.968571

9, 5, 1	0.5	64	500	0.968571

3.3.4 Epochs

The experimental results indicated that using 500 epochs is enough to get to a stabilized performance in accuracy with or without dropouts. The number of epochs can be reduced (e.g., 300) to get stabilized performance without dropouts.

3.3.5 Optimizer

As shown in the table below, the Adam optimizer works the best in terms of performance in accuracy. This should not be a surprise because Adam is an improvement of SGD.

Optimizer	Hidden Layers	Dropouts	Epochs	Batch Size	Accuracy
Adam	**9, 5, 1**	**0.5**	**500**	**16**	**0.974468**
SGD	9, 5, 1	0.5	500	16	0.971429
RMSprop	9, 5, 1	0.5	500	16	0.965714

4. Results

4.1 Model Evaluation and Validation

Similarly to [4], the dataset is split into two parts: 75% for training and the remaining 25% for testing.

To avoid the small dataset size issue, the original dataset of 699 samples is expanded to a new dataset of 1,880 samples by introducing normal distributed random noise in this capstone project.

As described before, similarly to the evaluation method practiced by Geekette [4], The prediction accuracy is used as the major evaluation metrics.

4.2 Justification

The prediction accuracy of the new supervised deep learning network is compared with the result of applying the open source scikit-learn out-of-box implementation of the Random Forest Classifier [6] with default settings to measure the relative performance.

The table below shows the summary of the comparison results of applying the 25% testing dataset to the trained Random Forest Classifier and the trained new deep learning network model.

Machine Learning Algorithm	Settings	Accuracy
scikit-learn Random Forest Classifier	Default	0.974468

Keras Deep Learning Network	epochs = 500
	batch_size = 16
		0.974468

Table 2: Summary of model testing results.

It can be seen in the table above that the performance of the new deep learning network is competitive with the Random Forest Classifier in accuracy. Both of them achieved a prediction rate of more than 98% in accuracy.

This performance of the new deep learning network outperformed the results reported by both Stahl [3] and Geekette[4].

5. Conclusion

5.1 Free-Form Visualization

. It can be seen from Figure 4 that the performance of the new deep learning model in accuracy is quickly stabilized to the maximum.

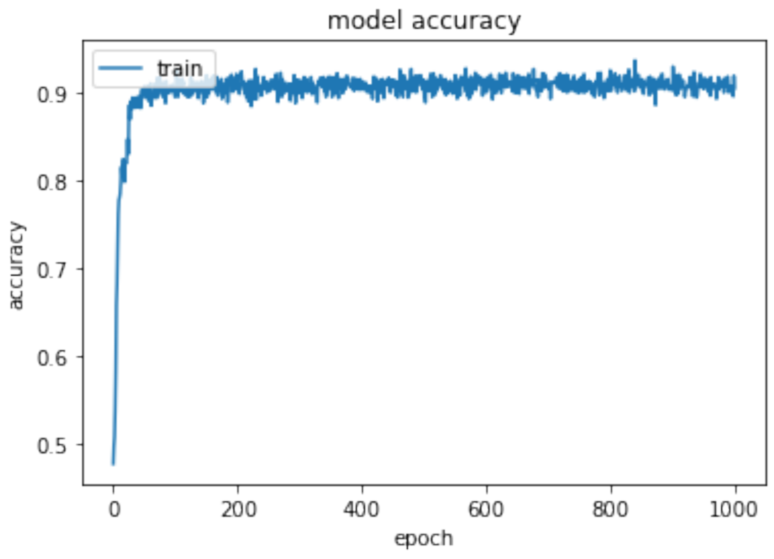

Figure 7: Accuracy history of the new deep learning network model in model training.

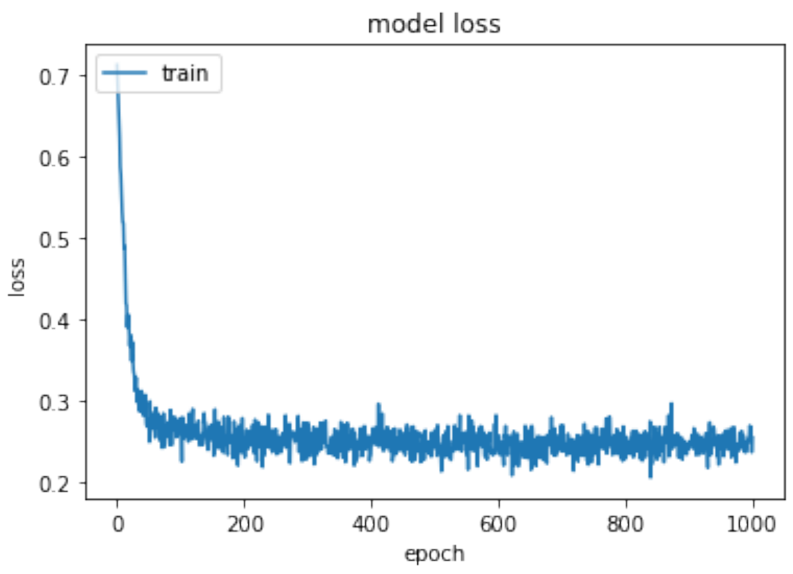

Figure 8: Loss history of the new deep learning network model in model training.

5.2 Reflection

This capstone project presented a new supervised deep learning network for the classification of the WBCD dataset [1][2] (see Section 2.3 for details).

A new data generation method was presented as well to expand the original dataset of 699 samples to a new dataset of 1,880 samples by introducing normal distributed random noise into existing data samples.

The missing data entries with the Bare Nuclei feature were replaced with zero, and then the data values were scaled into the range of [0, 1] for deep learning. The skewness problem with the dataset was significantly alleviated by taking squared root of the feature values.

The experimental results (see Section 4 for details) demonstrated that by using deep learning (with 3 hidden layers) and introducing Dropout layers and Sigmoid activation function in the final hidden layer, the new deep learning network is competitive with the traditional Random Forest Classifier in the classification of the WBCD dataset [1][2]. The performance of the new supervised deep learning network outperformed the performance results reported by both Stahl [3] and Geekette[4].

The new deep learning network was implemented in Python using well-known open source libraries such as Numpy, Pandas, scikit-learn, Keras, …, etc. The Jupyter notebook of this capstone project is provided in Github [8] for general reference.

5.3 Improvement

One common practice is to perform k-fold cross validation. This was not used in this capstone project for simplicity.

In addition, another possible improvement in the future is to identify and implement an automation mechanism to identify the best combination of hyper-parameters such as number of hidden layers, number of neurons in each hidden layer, dropouts and dropout rates, activation functions, batch size, epochs, …, etc.

The experimental results show that sometimes it is better to use a simple machine learning algorithm than a more advanced deep learning one. Typically deep learning is not needed with a simple dataset like the WSCD dataset [1][2].

References

[1] W. H. Wolberg, etc., "Wisconsin Breast Cancer Database (WBCD)": https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.names , January 8, 1991

[2] WBCD data file: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data, January 8, 1991

[3] K. Stahl, "Wisconsin Breast Cancer Diagnosis Deep Learning":

http://www.rpubs.com/kstahl/wdbc_ann, July 17, 2017

[4] D. Geekette, "Breast Cancer data: Machine Learning & Analysis":

http://rpubs.com/elena_petrova/breastcancer, November 25, 2016

[5] L. a comment, "Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic", Scientific Research, Vol.6 No.5, May 2013, http://www.scirp.org/journal/PaperInformation.aspx?PaperID=31887

[6] sk-learn Random Forest Classifier: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[7] Keras: The Python deep learning library: https://keras.io/

[8] Y. Zhang, Capstone Project in GitHub https://github.com/mlyuefeng/machine-learning/tree/master/capstone