Project 3 – Artificial Neural Networks and Deep Learning
CS548 / BCB503 / CS583 Knowledge Discovery and Data Mining - Fall 2019
Prof. Carolina Ruiz

Students: Bhoomi Patel and Srinarayan Srikanthan

	Classification	Regression
Dataset :		
• Dataset Description	/05	
• Data Exploration	/10	
• Initial Data Preprocessing (if any)	/05	
Code Description: Python libraries and functions, and/or your own code	/10	/10
Experiments:		
• Guiding Questions	/10	/10
• Sufficient & coherent set of experiments	/10	/10
• Objectives, Parameters, Additional Pre/Post-processing	/10	/10
• Presentation of results	/10	/10
• Analysis of individual experiments' results	/10	/10
Summary of Results, Analysis, Discussion, and Visualizations	/20	/20
Advanced Topic	/30	
Total Written Report	/210 = /100	

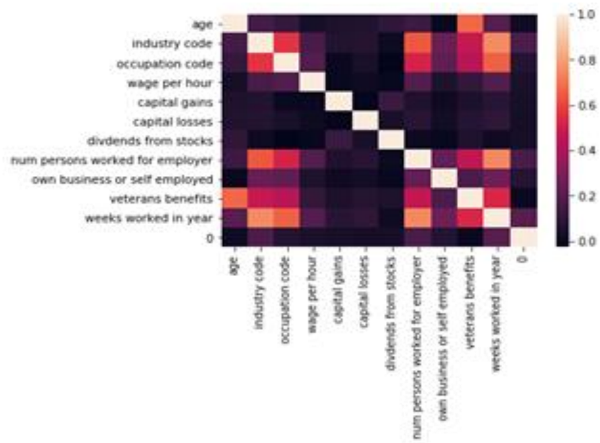
Dataset Description, Exploration, and Initial Preprocessing: (at most 1 page)

[05 points] Dataset Description: (e.g., dataset domain, number of instances, number of attributes, distribution of target attribute, % missing values, …)

The dataset is from the census bureau database to determine the income levels of a person. The number of instances is 199523, with 40 attributes considered for modelling. The target has two classes each having a distribution of 93.80% and 6.20% respectively. The following fields have missing values: State of previous residence(709-0.35%),migration code-change in msa(99696-49.96%),migration code-change in reg(99696-49.96%),migration code-change within reg(99696-49.96%),migration prev res in sunbelt(99696-49.96%), country of birth father(6713-3.36%), country of birth mother(6119-3.06%), country of birth self(3393-1.70%)

[10 points] Data Exploration: (e.g., comments on interesting or salient aspects of the dataset, visualizations, correlation, issues with the data, …)

The dataset has 33 Nominal attributes and 7 Continuous attributes .The dataset is an extensive one, with nearly 2 lakh instance representing the actual population. From the dataset it is observed that among numeric attributes capital gains has the highest standard deviation of 4697.53 and year has the lowest standard deviation of 0.5. It is also observed that capital losses has maximum number of zeros with a percentage of 98.04. By plotting the correlation matrix, we interpreted that *Capital Gains* column is highly correlated to the target variable *income*. The major issue with the dataset is the % of missing values it contains. Of the total columns that contained the missing values, majority columns has missing data almost equal to 50%. Another issue was the distribution of the income attribute, that was used as a classification predictor. It was clearly dominated by one class of values, making it difficult to develop a model to learn patterns in the weaker class.



[05 points] Initial data preprocessing, if any, based on data exploration findings: (e.g., removing IDs, strings, …)

Removed the entire column *instance weight*. Further there were missing values represented as '?' and the column hispanic Origin has values NA. These values were converted to categorical values using the following: df = df.replace('?', 'Missing'); df = df.fillna(df['hispanic Origin'].mode()[0])

Code Description: Python Libraries and Functions you used and what parameters you experimented with. (At most 1 page.)

[10 points] Classification:

Preprocessing Techniques for Classification: First, we used label encoding for *target* column using function *LabelEncoder()* and then, we *OneHotEncoded* all the nominal values using the function *pd.get_dummies(df) i.e.* OneHotEncoding with drop = 'first'. After encoding, all the variables where scaled using the *StandardScaler()* function.

ANNs: Libraries used:pandas, *sklearn.neural_network.MLPClassifier, sklearn.model_selection.KFold,sklearn.metrics,sklearn.metrics.accuracy_score, sklearn.metrics.precision_score, sklearn.metrics.recall_score, sklearn.metrics.auc_score.*

The MLPClassifier is invoked for each split of the k-folds with the following parameters: hidden_layer_sizes(the number of layers and number of nodes in each layer), activation(type of activation function for the hidden layer), solver(chosen based on the size of the dataset for weight optimization), alpha, batch_size(size of minibatches for stochastic optimizers), learning_rate(constant/adaptive/invscaling), random_state and momentum(if solver = 'sgd'). These parameters are varied depending on the guiding question and experimented.The performance of every experiment was evaluated using metrics provided by sklearn along with k-fold validation and obtain loss on each iteration using *clf.loss_*.

The loss function used for classification is log-loss. This is given by: $-\log P(yt|yp) = -(yt \log(yp) + (1 - yt) \log(1 - yp))$

[10 points] Regression:

Preprocessing Techniques for Regression: Used label encoding for column *education* using function *LabelEncoder()* and then, we *OneHotEncoded* all the nominal values using the function *pd.get_dummies(df) i.e.* OneHotEncoding with drop = 'first'. Used *sklearn.preprocessing.scale* to scale the attributes

ANNs: (if same as for classification above, just state so)

Libraries used: *pandas, numpy, sklearn.model_selection.K_Fold, sklearn.preprocessing.scale, sklearn.preprocessing.LabelEncoder, sklearn.neural_network.MLPRegressor, sklearn.metrics.mean_absolute_error, sklearn.metrics.mean_squared_error, sklearn.metrics.r2_score*

Using *KFold=20,* split the dataset into training and testing. For ea7ch split, do the following: use *MLPRegressor* using the *parameters hidden_layer_sizes* (the no. of hidden layers with nodes in each layer), *activation* (the type of activation for the hidden layer), *solver* (chosen based on the size of the dataset for weight optimization), *alpha* (L2 penalty), *batch_size* (size of mini batches for sgd), *learning_rate* (step-size for weight updation), *momentum* (if solver = 'sgd'), early_stopping, etc. Using *fit* function, fit the training dataset and then using *predict* function, compute y_pred for for the test dataset. Obtain loss on each iteration using *clf.loss_.* Compute correlation coefficient, mean_absolute_error and mean_squared error using the libraries mentioned above. Finally, compute the average of all the performance metrics.

The loss function used here is Squared-loss function. It is also known as Mean Square Error, which is square of difference between actual and predicted values

Note: We ran experiments with values of kFold ranging from 25 to 125 and there were negligible changes in the performance metrics. The only difference was that the time taken to build the model increased drastically with increase in kFold. Thus, we decided to perform experiments using k=20.

Guiding Questions

[10 points] Three Guiding Questions for the Classification Experiments: (at most 1/3 page)

1. Predict income based on capital gains, dividends from stocks, weeks per year and age.
2. Does the type and category of work determine a person's income.
3. Are income and demography of the person related?

[10 points] Three Guiding Questions for the Regression Experiments: (at most 1/3 page)

1. Determine age from a person's earnings.
2. Delineate age based on income, education and marital status
3. How old is a house-owner?

[40 points] Summary of Classification Experiments. Use k-fold cross-validation for a reasonable k, if possible. What k did you use? k =20.						
Guiding questions	**Pre-process**	**Parameters:** # of hidden layers, # of nodes in each hidden layer, activation function, "solver", learning rate, momentum, ...	**Performance metrics:** Accuracy, Precision, Recall, ROC Area, ... [specify which used]	**Time to build model**	**Analysis & observations about experiment, and interesting results**	**Calculated Average Loss**
1	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=1, # of nodes in each hidden layer=10, activation function='relu', solver=sgd, alpha=0.0001, batch size= 'auto', learning_rate='constant',(tol=1e-3)	Accuracy:94.40 Precision:74.47 Recall:17.45 ROC Area:58.43	3minutes 5seconds	With one layer the network was able to perform good, identifying both classes and the average loss over 20 iterations was low.Using tol=1e-3 reduced time and precision but not recall and loss.	15.87
1	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=2, # of nodes in each hidden layer=10,5 activation function='relu', solver=sgd, alpha=0.0001, batch size= 'auto', learning_rate='constant'	Accuracy:94.56 Precision:71.29 Recall:19.64 ROC Area:59.55	4minutes 19seconds	As the number of layers was increased the time to build the network increased but the accuracy and other parameters including average loss got better.	15.68
1	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=2, # of nodes in each hidden layer=10,5, activation function='relu', solver=sgd, alpha=0.0001, batch size= 'auto', learning_rate=adaptive	Accuracy:94.54 Precision:70.80 Recall:19.81 ROC Area:59.63	10minutes 7seconds	Changing the learning rate to adaptive had a drastic impact on the time to build the network with not any big difference in the remaining parameters.	15.61
2	OneHotEncoding, LabelEncoding, Scaling	#of hidden layers=3, #of nodes in each hidden layer=10,5,2 activation function =tanh, solver=sgd, alpha =0.0002, batch size=200, momentum =0.5 learning_rate ='invscaling',	Accuracy:93.76 Precision:29.85 Recall:2.41 ROC Area:50.09	10minutes 48seconds	The network with three hidden layers was not very good in finding the minority class and hence had a low recall and precision value, the high accuracy is attributed to the distribution of data.	18.20
2	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=3, # of nodes in each hidden layer=10,5,2, activation function=tanh, solver=adam, alpha=0.0002, batch size= 'auto', learning_rate=constant	Accuracy:93.82 Precision:54.73 Recall:3.25 ROC Area:51.86	8minutes 25seconds	By changing the learning rate from invscaling the time to build network reduced and changing the solver type provided better results in terms of precision and avg loss.	17.35
2	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=3, # of nodes in each hidden layer=10,5,2, activation function=tanh, solver=lbfgs, alpha=0.0002, batch size= 'auto', learning_rate='constant'	Accuracy:93.85 Precision:57.82 Recall:3.65 ROC Area:51.74	8minutes 6seconds	Modifying the solver to lbfgs and keeping the value of alpha at 0.0002 provided the best results out of the experiments that were run in all.	17.15

2	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=2, # of nodes in each hidden layer=10,5 activation function= logistic, solver=lbfgs, alpha =0.0001,batch size= 400, momentum =0.8 ,learning_rate = 'constant'	Accuracy:92.87 Precision:44.95 Recall:2.86 ROC Area:51.36	7minutes 15seconds	By reducing the number of hidden layers batch size and changing the activation function had a bad impact on performance though it marginally reduced the time.	17.18
3	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=2, # of nodes in each hidden layer=10,5 activation function=tanh, solver=adam, alpha=0.0002, batch size= 250, learning_rate ='constant'	Accuracy:93.78 Precision:8.33 Recall:1.61 ROC Area:50.0016	5minutes 12seconds	For this guiding question, varying any of the available parameters resulted in similar accuracy, but recall and precision was very low, revealing that there was no finite pattern between the parameters.	22.58
3	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=2, # of nodes in each hidden layer=10,5 activation function= tanh, solver=adam, alpha=0.0001, batch size= 500, learning_rate= constant	Accuracy:93.97 Precision:15.47 Recall:4.03 ROC Area:50.01	5minutes 42seconds	By changing the batch size to 500 and alpha value provided the best possible result for this guiding question considering all the other experiments by varying the parameters.	22.37

[20 points] Summary of Python Classification Results, Analysis, Discussion, and Visualizations (at most 1/2 page) 1. Analyze the effect of varying parameters/experimental settings on the results. 2. Analyze the results from the point of view of the dataset domain, and discuss the answers that the experiments provided to your guiding questions. 3. Comparison with regression results obtained in project 2.

1. From the experiments performed, it was evident that as a)hidden_layer_sizes: we increase the number of hidden layers, nodes in a layer or the learning rate as adaptive or invscaled the time required to construct the network increased. b)activation: The type of activation function used and the solver had a major role in developing the network. For the third guiding question it was *activation function= tanh, solver=adam* that was able to identify the minority class among data. Other combinations of solvers and activation functions were not able to do so. As time to build the network increased with more layers, performance also had a slight improvement.Having the batch_size small reduced time to construct a network. The average loss decreased when we fine tuned the parameters instead of the default ones.
2. From the domain perspective, there was no major relationship between income and demographic data nor the type and class of employment taken. By analysing the data, capital gains had a clear correlation with the target attribute ie., income. In this dataset, the number of people with an income of more than 50000 was very sparse. Finding patterns among them was a challenging task in which capital gains helped in identifying the sparse entries. This is the reason why the results of experiments on first guiding question are better than the other.
3. Comparing with the results of the experiments performed in the second project, the following can be inferred: 1) when there was no stopping condition given on the tree the precision and recall values were better than the neural network, and the accuracy was almost the same. 2) when stopping conditions were applied the precision and recall values dipped. The third guiding question being an exception where the neural network performed better in terms of the recall value. Depending on the features considered and its distribution different models/networks performed better. ZeroR outperformed with all metrics which should be attributed to the distribution of the data, except ROC as it identifies only one class.

[40 points] Summary of Regression Experiments Use k-fold cross-validation for a reasonable k, if possible. What k did you use? k = 20. *At most 1.5 pages.*						
Guiding questions	**Pre-process**	**Parameters:** # of hidden layers, # of nodes in each hidden layer, activation function, "solver", learning rate, momentum, ...	**Performance metrics:** Correlation Coefficient and Error Metric(s) [specify which used]	**Time to build model**	**Analysis & observations about experiment, and interesting results**	**Root Mean Square Error**
1	OneHotEncoding, Scaling	# of hidden layers=1, # of nodes in each hidden layer=10, activation function='relu', solver='adam', alpha=0.0001, batch size= 'auto', random_state=np.random	Loss = 0.47 Correlation Coefficient = 0.06 Mean Absolute Error = 0.78	1 min 39 secs	The model was able to give a good value for correlation coefficient, thus, stating that the attributes were highly correlated to the target attribute	0.97
1	OneHotEncoding, Scaling	# of hidden layers=3, # of nodes in each layer=10,5,7, activation function='relu', solver='sgd', alpha=0.0001, batch size='auto', learning_rate='invscaling', momentum=0.9	Loss = 0.48 Correlation Coefficient = 0.04 Mean Absolute Error = 0.80	1 min 56 secs	Here, though the attributes were highly correlated to target, the correlation coefficient decreased due to the value of the momentum. Thus, with the increase in friction i.e momentum, corr coeff decreases	0.98
1	OneHotEncoding, Scaling	# of hidden layers=2, # of nodes in each layer=10, activation function='identity', solver='lbfgs', alpha=0.0001, batch_size='auto'	Loss = 0.47 Correlation Coefficient = 0.05 Mean Absolute Error = 0.79	11 secs	As the no. of hidden layers increased, the corr coeff improved, thus stating that the no. of hidden layers play an important role in determining the metrics	0.97
2	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=2, # of nodes in each layer=10,5, activation function='tanh', solver='sgd', alpha=0.0001, batch_size=100, learning_rate='adaptive', momentum=0.5	Loss = 0.15 Correlation Coefficient = 0.69 Mean Absolute Error = 0.41	27 mins 29 secs	With learning rate = 'adaptive', the loss decreased drastically. This was because with every iteration the loss decreased and hence the model took so long to build	0.55
2	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=1, # of nodes in each layer=10, activation function = 'logistic, solver='lbfgs', alpha=0.0005, batch_size='auto', early_stopping=True	Loss = 0.15 Correlation Coefficient = 0.69 Mean Absolute Error = 0.40	6 mins 47 secs	The time taken to build this model was quite less compared to the above because of the solver and the activation function. Yet the metrics obtained were quite similar	0.55
2	OneHotEncoding, LabelEncoding, Scaling	# of hidden layers=3, # of nodes in each layer=5,5,2, activation function='relu', solver='adam', alpha=0.0007, batch_size=200, random_state=0	Loss = 0.15 Correlation Coefficient = 0.69 Mean Absolute Error = 0.41	7 mins 20 secs	With the increase in alpha there was a decrease in the variance, thus avoiding the model from overfitting the training data	0.55

3	OneHotEncoding, Scaling	# of hidden layers=4, # of nodes in each layer=10,5,2,2, activation function='identity', solver='adam', alpha=0.0001, batch_size='auto', beta_1=0.7	Loss = 0.22 Correlation Coefficient = 0.56 Mean Absolute Error = 0.50	2 mins 21 sec	Using beta_1=0.7; the exponential decay rate, the model trains faster and produces results quicker compared to the rest	0.66
3	OneHotEncoding, Scaling	# of hidden layers=2, # of nodes in each layer=10,2, activation function='logistic', solver='lbfgs', alpha=0.0005, batch_size=200, random_state=np.random	Loss = 0.21 Correlation Coefficient = 0.56 Mean Absolute Error = 0.50	1 min 54 secs	Here, as the batch_size was set to 200, the time taken to build the model was less as compared to the others, because the weights are updated after each propagation	0.65
3	OneHotEncoding, Scaling	# of hidden layers=5, # of nodes in each layer=3,3,2,3,2, activation function='relu', solver='sgd', alpha=0.0002, batch_size='auto', early_stopping=True	Loss = 0.22 Correlation Coefficient = 0.56 Mean Absolute Error = 0.50	5 mins 23 secs	early_stopping=True uses the validation data to score the model and hence avoids overfitting by returning the best parameters	0.66

[20 points] Summary of Python Regression Results, Analysis, Discussion, and Visualizations (at most 1/2 page) 1. Analyze the effect of varying parameters/experimental settings on the results. 2. Analyze the results from the point of view of the dataset domain, and discuss the answers that the experiments provided to your guiding questions. 3. Comparison with regression results obtained in project 2.

1. Based on the various parameters used, following was our observation: a)hidden_layer_sizes: As the size of hidden layer increases, the performance metric increases but the time taken to build the model also increases. b) activation and solver: These two together are an important criteria to determine the model. With sgd as solver and adaptive as activation, the accuracy of the model increased. c) batch_size: the network trains faster with mini batches. d) early_stopping: early_stopping helps avoid overfitting the model. e) alpha: Increase in alpha helps to control the high variance of the model

2. Guiding Question 1: We found that wage per hour, income and weeks worked in a year would play a major role in determining the person's age. However, the results obtained weren't that good, thus, stating that there is no direct relation between a person's age with the income he earns by working for a particular weeks in a year.

Guiding Question 2: This question produced the best correlation coefficient, thus signifying that education, marital status and income played a very critical role in determining a person's age.

Guiding Question 3: Further, we decided to compute the age of a person who owns a house i.e is a householder. The performance metrics obtained were moderate and implied that there was some relationship between the attribute detailed household summary in household = Householder and the target attribute age.

3. By comparing results with project 2**,** following was inferred; Guiding Question 1: With one hidden layer, solver='adam' and activation='relu', the model predicted the same results as the DecisionTreeRegressor. On comparing with zeroR, the results were somewhat closer. Guiding Question 2: The performance metrics obtained with solver='sgd' and learning_rate='adaptive' were quite similar to this model and hence, the neural network was able to predict the same as the DecisionTreeRegressor. Guiding Question 3: With alpha=0.005 and batch_size = 100, similar results were obtained, thus implying that both model correctly avoided high variance and hence ultimately prevented the model from overfitting the training data.

Advanced Topic (AT MOST 1 PAGE): Conditional Generative Adversarial Networks (cGANs)

[7 points] List of sources/books/papers used for this topic (include URLs if available). *Provide full references (authors, title, where published, year, …).*

- Jon Gauthier, Conditional generative adversarial networks for convolutional face generation, Technical report, March 2015 (https://www.foldl.me/uploads/2015/conditional-gans-face-generation/paper.pdf)
- Mehdi Mirza, Simon Osindero, Conditional Generative Adversarial Nets, arXiv, Cornell University, 6 Nov 2014 (https://arxiv.org/pdf/1411.1784.pdf)
- Connor Shorten, Must - Read Papers on GANs, TowardsDataScience blog, Mar 4 2019 (https://towardsdatascience.com/must-read-papers-on-gans-b665bbae3317)

[20 points] In your own words, provide an in-depth, yet concise, description of your chosen topic. Make sure to cover all relevant data mining aspects of your topic. *Your description here should be comprehensive and in-depth (it should reflect work at the graduate level).*

Generative Adversarial Network are deep Neural Nets that consists of two 'adversarial' models viz. Generator model G which tries to capture the distribution of data and generates new data instances and the Discriminator model D which estimates the probability if its from the training sample or from G. To learn a generator distribution over data x, a mapping function from is build from noise distribution and data spaces. Both G and D are trained simultaneously, the values of G and D are augmented to minimize the value of log(1 − D(G(z)) and maximize log D(x) respectively. The mapping function is represented using the following equation:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Conditional Generative Adversarial Networks (cGANs) are an extension of GANs wherein an external condition *y* is applied to both the Discriminator and the Generator. This further assists in building models with different contextual information by applying different conditions and helps the generator in generating fake samples with a specific condition. For eg: with the help of noise data we could generate face of a person with sunglasses. This makes cGAN not completely unsupervised as we need some class labels to guide them. In the objective function for cGANs, the generator has prior noise and condition *y* are combined and for the discriminator the data x and condition *y* are presented as inputs. In the discriminator x and y are presented as inputs to a discriminative function and in the generator the prior input noise z, and y are combined in joint hidden representation. This provides the network a headstart in what to look for. The objective function is represented as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

In conclusion, cGANs have lot of potential for interesting and useful applications. Various experiments performed by the authors in the above mentioned papers have proved that cGANS are way more effective than conventional model and that the conditional information y could be used to deterministically control the output of the generator; which further implied that cGANs could be used to generate images from spoken text or handwritten words.

[3 points] Describe how this topic relates to deep learning and the material covered in this course.

In this course, we discussed how deep learning uses hierarchy of neural networks to imitate our brain. Since, cGAN; which is a deep learning and unsupervised (not completely) machine learning technique in GAN also consists of Multilayer Perceptrons (MLP) which has various applications, this topic is highly related to Deep Learning.

Authorship: The initial Data Exploration was done by Bhoomi whereas visualizing the dataset to obtain correlation was done by Sri. Both of the members came up with different guiding questions and both built their code, ran experiments and then based on the results, merged the code to obtain the best output for the guiding questions. Mutually decided which would be the best part of the model to incorporate in the report and it was GAN concept that motivated Sri to choose the advanced topic. Based on the topic, Bhoomi came up with the concept of Conditional GANs.