S3710646 Xinhong Chen S3643938 Yi Weng

Data Mining Assignment 2

PART 1: CLASSIFICATION WITH NEURAL NETWORKS

1. In this case, we have a 15 attributes 606 data sets. We make it into 29 input units and 2 output units (in this case we make num {<50,>50_1} as output). There are 7 attributes as Numeral data (age, weight, trestbps, chol, thalach, oldpeak, and ca). The others are 8 Nonnumeric data attributes, the data was taken like for example the inputs are {male, female} then it was encoded male as {1 0} and female as {0 1}. And we encode to {0,0,0} as missing when the value meets "?" in thal {fixed_defect,normal,reversable_defect}. And By observing the distribution of Numeral data by attributes, we find oldpeak is following linearly decreasing, ca is more like an Ordinal Numeric data attribute so we encode it like below table. The others like following Gaussian distribution, thus we encode them with oldpeak into weka normalise.

ca					
Value	1	2	3	4	?(missing)
Encoding	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1	0 0 0 0
thal					
value	fixed_defect	normal	reversable_defect	?(missing)	
Encoding	1 0 0	0 1 0	0 0 1	0 0 0	

2. The numeric attributes are normalised by using weka, the related script is in Appendix.
3. We use WTA strategy which means the greatest value of output will dominate the result. And with default, values are h = 0.0, l = 0.0. When a the greatest value of an output unit for a pattern, this unit is the biggest one in the teaching output as well, this pattern is classified correctly. When this unit is not the biggest one, it is classified incorrectly.
4. Through experiments, we found that the training error rate reduced rapidly at the beginning, and eventually it remains in a certain level. But testing error often decreased slightly in a short epochs, and it will increase while keeping training (overfitting becomes larger). Obviously, the No.1 result is the best results of the 5 runs with the least value of overfitting of 0.0792 and related low testing error as 8.24%. The other 4 runs have relatively big overfitting (around 0.13-0.16) for the number of result between 0 and 1.

Run No	Architecture	Parameters	Train MSE	Train Error %	Epochs	Test MSE	Test Error %	Overfitting
1	29-10-2	lr=0.2	0.0385	1.9	100	0.1177	8.24	0.0792
2	29-10-2	lr=0.2	0.1803	0.99	100	0.99	9.89	0.157
3	29-10-2	lr=0.2	0.0431	1.98	100	0.1825	12.64	0.1394
4	29-10-2	lr=0.2	0.0122	0.33	100	0.1767	9.34	0.1645
5	29-10-2	lr=0.2	0.0233	0.99	100	0.1601	10.44	0.1368

5. We have experimented 5 different numbers of hidden nodes. It seems that when the hidden nodes are 7 (successful) it is the best comparing with others. Because it has the lowest overfitting and a proper test error. Nodes 5 has large test error which cannot be reduced to a suitable level (unsuccessful). When the number of nodes is over 10, it has already got the most accuracy at the beginning. Therefore, 7-10 nodes might be the right number.

Run No	Architecture	Parameters	Train MSE	Train Error %	Epochs	Test MSE	Test Error %	Overfitting
1	29-5-2	lr=0.2	0.0476	1.98	50	0.1826	10.99	0.1350
2	29-7-2	lr=0.2	0.0407	1.65	50	0.1436	8.74	0.1029
3	29-10-2	lr=0.2	0.0135	0.66	50	0.1666	9.89	0.1531
4	29-12-2	lr=0.2	0.0369	1.65	50	0.1621	9.34	0.1252
5	29-15-2	lr=0.2	0.0484	1.98	50	0.1649	9.89	0.1164

6. By observing the results, the worse result is when the value of the learning rate is 0.5. At that time, we found that the error rate for training sets decreased rapidly (It is learning too much). In contrast, the error rate for testing sets increased slowly or remained. As a result, 0.2 or 0.3 has better result with the lower error rate and overfitting comparing with others.

Run No	Architecture	Parameters	Train MSE	Train Error %	Epochs	Test MSE	Test Error %	Overfitting
1	29-10-2	Ir=0.1	0.0437	2.31	50	0.1678	10.38	0.1241
2	29-10-2	Ir=0.15	0.0366	1.65	50	0.1720	10.93	0.1354
3	29-10-2	Ir=0.2	0.0298	1.32	50	0.1533	7.65	0.1235
4	29-10-2	Ir=0.3	0.0540	2.97	50	0.1700	10.93	0.1160
5	29-10-2	Ir=0.5	0.0479	2.31	20	0.1957	12.02	0.1478

7. We experimented the learning rate from 0.1 to 0.8 with the value of momentum (0.1 – 0.8), and record 5 best result in the table. By observing the error graphic with different parameters. We found that the training error will fluctuate when the value of momentum is a large number (like 0.8), and overfitting was increasing. Especially, when the value of learning rate is also a large number. Therefore, we tried to increase the value of Ir and decrease the value of momentum. Then we found the best result with Ir=0.5 and Momentum=0.3.

Run No	Architecture	Parameters	Train MSE	Train Error %	Epochs	Test MSE	Test Error %	Overfitting
1	29-10-2	Ir=0.1 Momentum=0.5	0.0413	2.31	20	0.1850	11.54	0.1437
2	29-10-2	Ir=0.2 Momentum=0.5	0.0439	2.13	20	0.1820	12.09	0.1381
3	29-10-2	Ir=0.3 Momentum=0.4	0.0744	0.96	20	0.2419	15.38	0.1675
4	29-10-2	Ir=0.5 Momentum=0.3	0.0845	4.62	20	0.1961	12.09	0.1116
5	29-10-2	Ir=0.8 Momentum=0.1	0.0899	4.95	20	0.2264	12.64	0.1365

8. Although we ran the training again and again to touch the most accuracy of training sets. We did not get the most accuracy for testing sets, the error rate might be higher than the start point. And the overfitting has the largest gap from our other results.

Run No	Architecture	Parameters	Train MSE	Train Error %	Epochs	Test MSE	Test Error %	Overfitting
1	29-10-2	Ir=0.2	0.0096	0.33	100	0.2482	13.19	0.2386

9. Comparing with three different classifiers, Multilayer Perceptron absolutely has the most accuracy testing result and the lowest MSE overfitting. We calculated the Root mean squared error of J48 and Multilayer Perceptron to MSE and compared with neural.

Run No	Architecture	Parameters	Train MSE	Train Error %	Epochs	Test MSE	Test Error %	Overfitting
1	29-10-2	Ir=0.2	0.0385	1.9	100	0.1177	8.24	0.0792

Run No	classifier	Parameters	Train MSE	Train Error %	Test MSE	Cross Validation %	Overfitting (MSE)
1	J48	-C 0.18 –M 1	0.0302	3.30	0.0661	7.26	0.0358
2	Multilayer Perceptron	-L 0.3 –M 0.2 –N 500 –V 0 –S 0 –E 20 –H a	0.0165	1.65	0.0304	2.97	0.0139

PART 2: NUMERIC PREDICTION WITH NEURAL NETWORKS

1. In this case, we have a 15 attributes 606 data sets. We split sets for 50% training, 20% validation and 30% testing. And we make it into 30 input units and 1 output units. We use the same method as the previous question encoding the **Numeral** and **Nonnumeric** data except **weight** attribute. The weight data is encoded by the formula of **y = 0 + (x – min*0.9)*(1 – 0)/ (max*1.1-min*0.9)** to range of 0-1 with Java Programming.
2. The numeric attributes are normalised by using weka, weight data is encoded by the Java Programming. The related script is in Appendix.
3. Here we use MSE to determining the relatively high weight to large errors. By observing, the graphics decreased more slowly than previous question. And the drop of testing error can be captured easily. Therefore, we got the best overfitting as 0.00072 out of 5 runs. All 5 results have the slightly small overfitting.

Run No	Architecture	Parameters	Train MSE	Epochs	Test MSE	Overfitting
1	30-5-1	lr=0.2	0.0024	100	0.00328	0.00084
2	30-5-1	lr=0.2	0.0026	100	0.00353	0.00093
3	30-5-1	lr=0.2	0.0027	100	0.00411	0.00141
4	30-5-1	lr=0.2	0.0028	100	0.00362	0.00082
5	30-5-1	lr=0.2	0.0027	100	0.00342	0.00072

4. We have experimented 4 different numbers of hidden nodes (3, 5, 7, 10). It seems that when the hidden nodes are 5, it is the best comparing with others. Because it has the lowest overfitting as 0.00072 (successful). When the number of nodes is 10, the best is almost at the start point. And 3 nodes often need more epochs to generate better result. Therefore, 5 nodes might be the right number.

Run No	Architecture	Parameters	Train MSE	Epochs	Test MSE	Overfitting
1	30-3-1	lr=0.2	0.0025	100	0.00425	0.00175
2	30-5-1	lr=0.2	0.0027	100	0.00342	0.00072
3	30-7-1	lr=0.2	0.00309	100	0.00490	0.00181
4	30-10-1	lr=0.2	0.00245	100	0.00340	0.00095

5. By observing the results, the worse result is when the value of the learning rate is 0.5. At that time, this model almost got the best training at the beginning. Moreover, when value of learning rate from 0.1 to 0.3, we can do more epochs to get better overfitting result.

Run No	Architecture	Parameters	Train MSE	Epochs	Test MSE	Overfitting
1	30-5-1	lr=0.1	0.0027	100	0.0035	0.0008
2	30-5-1	lr=0.15	0.0027	100	0.00342	0.00072
3	30-5-1	lr=0.2	0.0027	100	0.00342	0.00072
4	30-5-1	lr=0.3	0.0025	100	0.00323	0.00068
5	30-5-1	lr=0.5	0.00256	100	0.00400	0.00144

6. We experimented the learning rate from 0.1 to 0.8 with the value of momentum (0.1 – 0.8), and record 5 best result in the table. We found that both the training error and validation error will decrease modestly while keeping training with the small number of learning rate (0.1) and Momentum (0.1). When the value of momentum or the value of learning rate increases, the drop of error become sharp, and the training will be finished quickly. Especially, when the value of learning rate is also a large number (0.5). Therefore, we tried to increase the value of momentum and decrease the value of Ir. Then we found the best result with Ir=0.1 and Momentum=0.7.

Run No	Architecture	Parameters	Train MSE	Epochs	Test MSE	Overfitting
1	30-5-1	lr=0.1 Momentum=0.1	0.0027	20	0.00336	0.00066
2	30-5-1	lr=0.1 Momentum=0.7	0.0024	20	0.00283	0.00043
3	30-5-1	lr=0.3 Momentum=0.3	0.0026	20	0.00341	0.00081
4	30-5-1	lr=0.5 Momentum=0.1	0.0025	20	0.00396	0.00146

7. Although we ran the training again and again to touch the most accuracy of training sets with 4 epochs. We did not get the more accuracy for testing sets. And the overfitting has the larger gap from our other results.

Run No	Architecture	Parameters	Train MSE	Epochs	Test MSE	Overfitting
1	30-5-1	lr=0.2	0.00256	100	0.00366	0.00110

8. We use java programming to reverse the weight data to raw range of kg and calculate the mean absolute error. In this case, M5P is the best classifier of these three methods with cross validation. The neural classifier is better than Multilayer Perceptron in this case. It has the similar testing MAE and lower overfitting. All of three classifiers can be acceptable, because the testing MAE can be acceptable with around the value of 2-4 in the range of samples of 45.088kg to 108.979kg. Their overfitting is under the suitable scope.

Run No	Architecture	Parameters	Train (MAE)	Epochs	Test (MAE)	Overfitting (MAE)
1	30-5-1	lr=0.2	3.4250	100	3.6782	0.2532

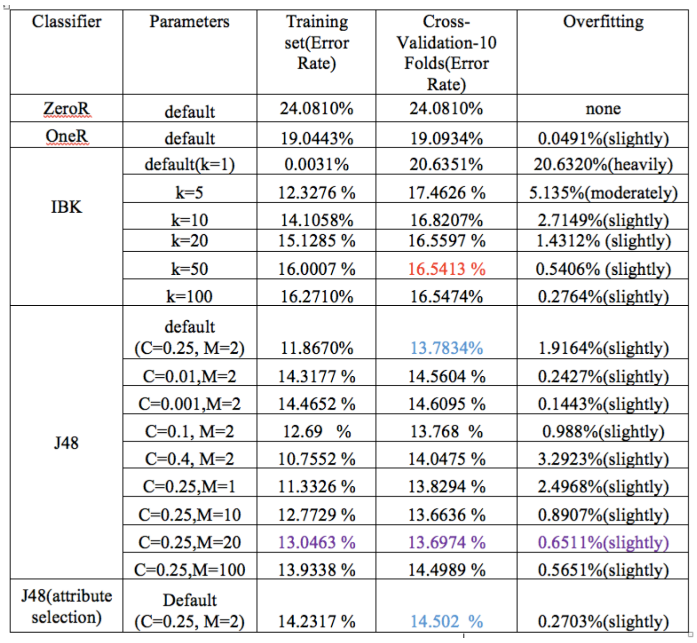
Run No	classifier	Parameters	Train (MAE)	Cross Validation (MAE)	Overfitting (MAE)
1	M5P	–M 4	2.4758	2.5079	0.0321
2	Multilayer Perceptron	-L 0.3 –M 0.2 –N 500 –V 0 –S 0 –E 20 –H a	2.8603	3.3577	0.4974

Part3:

The file we used is data/arff/UCI/adult.arff.

Aim: Look for any golden nuggets in adult file using classification, clustering, association finding, attribute selection and visualization.

Classification:



Classifier	Parameters	Training set(Error Rate)	Cross-Validation-10 Folds(Error Rate)	Overfitting
ZeroR	default	24.0810%	24.0810%	none
OneR	default	19.0443%	19.0934%	0.0491%(slightly)
IBK	default(k=1)	0.0031%	20.6351%	20.6320%(heavily)
	k=5	12.3276 %	17.4626 %	5.135%(moderately)
	k=10	14.1058%	16.8207%	2.7149%(slightly)
	k=20	15.1285 %	16.5597 %	1.4312% (slightly)
	k=50	16.0007 %	16.5413 %	0.5406% (slightly)
	k=100	16.2710%	16.5474%	0.2764%(slightly)
J48	default (C=0.25, M=2)	11.8670%	13.7834%	1.9164%(slightly)
	C=0.01,M=2	14.3177 %	14.5604 %	0.2427%(slightly)
	C=0.001,M=2	14.4652 %	14.6095 %	0.1443%(slightly)
	C=0.1, M=2	12.69 %	13.768 %	0.988%(slightly)
	C=0.4, M=2	10.7552 %	14.0475 %	3.2923%(slightly)
	C=0.25,M=1	11.3326 %	13.8294 %	2.4968%(slightly)
	C=0.25,M=10	12.7729 %	13.6636 %	0.8907%(slightly)
	C=0.25,M=20	13.0463 %	13.6974 %	0.6511%(slightly)
	C=0.25,M=100	13.9338 %	14.4989 %	0.5651%(slightly)
J48(attribute selection)	Default (C=0.25, M=2)	14.2317 %	14.502 %	0.2703%(slightly)

ZeroR, as the simplest classification method, relies on the target and ignores all predictors. It determines baseline performance as a benchmark for other classification methods. It predicts the majority class. ZeroR gives us 24.081% as the error rate and none overfitting. Therefore, from Classifier output, we can tell the majority class is income<=50k, so most of the adults' income less than 50k.

By running **OneR** with default parameter, we found in Adult file, the best predictor, minimum-error attribute, is Capital-gain. However, the overfitting rate is 0.0491%, error rate for training set and error rate for cross-validation is around 19%. OneR only gives us a slightly better performance when comparing with ZeroR.

We did several runs with changing the value of K (from 0 to 100) in **IBK**, while K increases , the overfitting rate decreases. The error rate in cross-validation does not follow any order. Then it is not appropriate to say the larger k is, the better performance we get. Because if K is too large, the nearest neighbors around the test sample may cross class boundaries. The class of the test set is determined by the majority of the class of the nearest neighbors. It is important to get the appropriate k value to minimize the error. In adult file, the noisy points in the training set also influence the performance of IBK. We can only conclude that IBK with K=50 has a better performance than OneR.

We tried running AttributeSelection(CfsSubsetEval, BestFirst-5) before running J48, it gives us 5 attributes, Education2, Marital_status, Relationship, Capital-gain and Capital-loss. However, comparing the output of running J48 with default parameters with Full attributes, the subset attributes gives us higher error rate.

In **J48**, when M=2 and by changing the number of C (from 0.001 to 0.4), default parameter gives us better accuracy. When we run J48 with parameter M ranging from 1 to 100 and C=0.25, the overfitting rate decreases when the parameter M increases. However, there is no significant pattern between error rates. C is the confidence factor, it determines how aggressive the pruning process will be. So smaller value induce more pruning. It can affect classifier performance significantly. M determines what the minimum number of observations are allowed at each leaf of the tree. The combination of C=0.25 and M=20 give us the best performance by elevating error rate and overfitting. We can find several golden nuggets by visualizing the tree:

1. If people's capital-gain above 6849, then most likely their income is above 50k. This rule is quite reasonable.

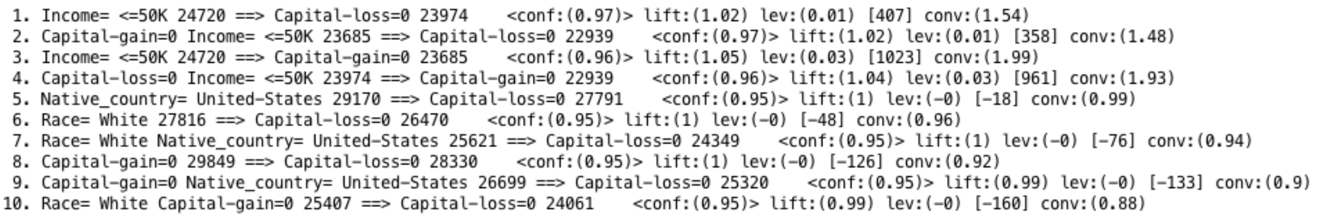
2. One interesting golden nugget is that, for people whose Capital-gain is <= 6849, their income relates to Marital_status. If they are divorced, or married-spouse-absent, or separated, most likely their income <= 50k.
3. For people whose Capital-gain <=6849 and have never-married, education2 and Age also helps to predict people's income. If their Education2 level is <= year 12, most likely their income is <= 50k.
4. If their Marital_status is Married-civ-spouse as well as Capital-gain<=6849, for people whose Capital-loss is between 1762 and 1980, most likely their income is above 50k.
5.Attribute Fnlwgt, Race and Native_country does not have a strong correlation with Income. Actually, attribute Fnlwgt can be deleted, because it is ambiguous and cannot be interpreted. After removing the Fnlwgt, we found the accuracy rate has been only slightly raised (around 0.3%).

Clustering:

There is no significant difference when we run **SimpleKMeans** and **EM** with Fnlwgt and without Fnlwgt. However, we just delete this attribute for easy understanding and analysis.
EM with default parameters generated two clusters, cluster 0 has 87% of the total instances, cluster 1 has 13%. Most people in Cluster 0 have income <= 50k. Mainly, their Marital_status is Never-married, their Occupation is Adm-clerical and Other-service, their relationship is Not-in-family, their Capital-gain and Capital-loss both low. Their Age, Education, Race, Sex, Hrs_week and Native_country are not conclusive for Cluster 0 and 1.
By running **SimpleKMeans** with default NumofCluster=2, cluster 0 has 42% of the total instances, cluster 1 has 58%. By observing mean, standard deviation, we found that the value for attribute Age, Worldclass, Race, Sex, Hrs_week, Captial_gain, Capital_loss and Native_country have significant overlap between two clusters, therefore we run several different values of NumofCluster with SimpleKMeans algorithm in order to get better clusters and useful golden nuggets. When NumofCluster =5, the golden nuggets we found are, in cluster 1, 99%(which is 5253 instances) of people's income <= 50k. The attributes could help with clustering includes, mainly, Some-college for Education 1, Never-married for Martial_status, Other-service for Occupation, Own-child for Relationship. Therefore, for people with these values in each attributes, most likely their income<=50k.
We also run clusterers without "class" attribute to elevate the results, in this case, we ignored attribute Income. SimpleKMeans with NumofCluster=2, seed =10 gives us two clusters, cluster 0 has 51% instances, cluster 1 has 49% instances. EM with default parameters also generates two clusters by its complex algorism with 87% of instances in Cluster 0 and 13% of instances in Cluster 1.
Then we compared Incorrectly clustered instances by running Classes to clusters evaluation, Incorrectly clustered instances in SimpleKMeans is 34.5198 % and it is 21.667 % in EM. Therefore, EM is more reliable. However, in EM, we found that there are only 2450 instances has been correctly clustered into Cluster 1(>50k), 5391 instances are incorrectly clustered into Cluster 1(>50k). Thus, the only golden nugget we can get by observing EM output is that people who never-married, work as Adm-clerical, Other-service, Not-in-family relationship, lower Capital-gain and lower Capital-loss most likely have Income <= 50k. In addition, there is no significant difference in Cluster 0 and Cluster 1 for attribute Education 1, Education 2, Race, Sex, Hrs_week and Native_country due to mode and overlap. Using two ways(with "class" attribute and without "class" attribute) to analysis clusters we get same golden nuggets, therefore, the golden nuggets we get is very reliable.

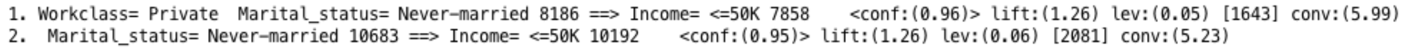
Association finding:

After using "NumericToNominal" filter to preprocess data, we run association finding without Fnlwgt. Best 10 rules we found as following:



```
 1. Income= <=50K 24720 ==> Capital-loss=0 23974    <conf:(0.97)> lift:(1.02) lev:(0.01) [407] conv:(1.54)
 2. Capital-gain=0 Income= <=50K 23685 ==> Capital-loss=0 22939    <conf:(0.97)> lift:(1.02) lev:(0.01) [358] conv:(1.48)
 3. Income= <=50K 24720 ==> Capital-gain=0 23685    <conf:(0.96)> lift:(1.05) lev:(0.03) [1023] conv:(1.99)
 4. Capital-loss=0 Income= <=50K 23974 ==> Capital-gain=0 22939    <conf:(0.96)> lift:(1.04) lev:(0.03) [961] conv:(1.93)
 5. Native_country= United-States 29170 ==> Capital-loss=0 27791    <conf:(0.95)> lift:(1) lev:(-0) [-18] conv:(0.99)
 6. Race= White 27816 ==> Capital-loss=0 26470    <conf:(0.95)> lift:(1) lev:(-0) [-48] conv:(0.96)
 7. Race= White Native_country= United-States 25621 ==> Capital-loss=0 24349    <conf:(0.95)> lift:(1) lev:(-0) [-76] conv:(0.94)
 8. Capital-gain=0 29849 ==> Capital-loss=0 28330    <conf:(0.95)> lift:(1) lev:(-0) [-126] conv:(0.92)
 9. Capital-gain=0 Native_country= United-States 26699 ==> Capital-loss=0 25320    <conf:(0.95)> lift:(0.99) lev:(-0) [-133] conv:(0.9)
10. Race= White Capital-gain=0 25407 ==> Capital-loss=0 24061    <conf:(0.95)> lift:(0.99) lev:(-0) [-160] conv:(0.88)
```

We can see all the Confidence value are above 0.9%. The confidence of the best top 1 rule is 0.97. It can be interpreted as the 97% of the occurrence for income <=50k, their Capital_loss is 0. After we remove redundancies, other rules we found is that, people whose income<=50 have 0 Capital_gain with confidence level of 96%.
However, if we would like to find the association rule for Income, we need to try different attributes. Using Attribute Workclass and Marrital_status, we found two best rules:



```
1. Workclass= Private  Marital_status= Never-married 8186 ==> Income= <=50K 7858    <conf:(0.96)> lift:(1.26) lev:(0.05) [1643] conv:(5.99)
2.  Marital_status= Never-married 10683 ==> Income= <=50K 10192    <conf:(0.95)> lift:(1.26) lev:(0.06) [2081] conv:(5.23)
```

The best rule can be interpreted as that the 96% of the occurrence of Workclass=Private and Martial_status= Never-married, their Income level<=50k. In addition, for people who never-married with 0.91 confidence that they have Income<=50K.

Attribute selection:

As we mentioned earlier in Classification section, Attribute selection is not particularly useful if the number of attributes is small. It has been proved in Classification section. We get best five attributes by running CfsSubsetEval as the Attribute Evaluator and BestFirst (N-5) as the Search Method and then run J48(default setting), the error rate even slightly larger than using full set attributes. In adult file, there are only 15 attributes. Attribute selection can be very effective when there are hundreds or thousands of attributes. So we won't run any other Attribute Selections for adult file.

Visualization:

By visualization, we found the following golden nuggets:
1.Most farmers' income is below 50k and for people who works 75 to 99 hours per week, farmer occupied higher percentage.
2. For people who have not married regardless working hours weekly, their income most likely less than or equal to 50k.
3. People who have higher education level will also have higher chance of achieving income above 50k.
4. For occupation, Adm-Exec and Professional has higher chance to achieve income above 50k.
5. People who have higher capital gain will have higher chance of getting income above 50k.

Appendix:

PART1:

```
#!/bin/bash
TRAIN=303
VALID=121
TEST=182

tail -n +20 heart-v1-pattern1.arff |\
  fgrep -v "%" | shuf |\
  sed -e"s/,/ /g"> temp1.csv

# The training file
/bin/echo "SNNS pattern definition file V3.2"  >heart-v1-train.pat
/bin/echo "generated at Mon SEP 25 15:58:23 2018"  >>heart-v1-train.pat
/bin/echo ""  >>heart-v1-train.pat
/bin/echo ""  >>heart-v1-train.pat
/bin/echo "No. of patterns : $TRAIN"  >>heart-v1-train.pat
/bin/echo "No. of input units : 29"  >>heart-v1-train.pat
/bin/echo "No. of output units : 2"  >>heart-v1-train.pat

head -$TRAIN temp1.csv  >temp-heart-v1-train.csv


cut -d' ' -f1,2,3,4 temp-heart-v1-train.csv |\
  sed -e "s/female/0 1/g" |\
  sed -e "s/male/1 0/g" |\
  sed -e "s/atyp_angina/0 0 0 1/g" |\
  sed -e "s/asympt/0 1 0 0/g"|\
  sed -e "s/non_anginal/0 0 1 0/g" |\
  sed -e "s/typ_angina/1 0 0 0/g" >>temp-1234.csv

cut -d' ' -f5,6,7,8,9 temp-heart-v1-train.csv |\
  sed -e "s/left_vent_hyper/1 0 0/g" |\
  sed -e "s/st_t_wave_abnormality/0 0 1/g" |\
    sed -e "s/normal/0 1 0/g"|\
  sed -e "s/t/1 0/g" |\
  sed -e "s/f/0 1/g" >>temp-56789.csv


cut -d' ' -f10,11,12 temp-heart-v1-train.csv |\
  sed -e "s/no/1 0/g" |\
  sed -e "s/yes/0 1/g" |\
  sed -e "s/down/1 0 0/g" |\
  sed -e "s/flat/0 1 0/g"|\
  sed -e "s/up/0 0 1/g" >>temp-101112.csv

cut -d' ' -f13 temp-heart-v1-train.csv |\
  sed -e "s/a/1 0 0 0/g"|\
  sed -e "s/b/0 1 0 0/g"|\
  sed -e "s/c/0 0 1 0/g"|\
  sed -e "s/d/0 0 0 1/g"|\
  sed -e "s/?/0 0 0 0/g" >>temp-13.csv

cut -d' ' -f14,15 temp-heart-v1-train.csv |\
  sed -e "s/fixed_defect/1 0 0/g" |\
  sed -e "s/normal/0 1 0/g"|\
  sed -e "s/reversable_defect/0 0 1/g" |\
  sed -e "s/?/0 0 0/g"  |\
sed -e "s/<50/1 0/g" |\
sed -e "s/>50_1/0 1/g" >>temp-1415.csv



paste -d, $FILE temp-1234.csv temp-56789.csv temp-101112.csv temp-13.csv temp-1415.csv |\
sed -e "s/,/ /g" |tr -d "\r">> heart-v1-train.pat
rm -r temp-1234.csv temp-56789.csv temp-101112.csv temp-1415.csv temp-13.csv temp-heart-v1-
train.csv

#cut EVERY attributes of the data

# The validation file
/bin/echo "SNNS pattern definition file V3.2"  >heart-v1-valid.pat
/bin/echo "generated at Mon SEP 25 15:58:23 2018"  >>heart-v1-valid.pat
/bin/echo ""  >>heart-v1-valid.pat
/bin/echo ""  >>heart-v1-valid.pat
/bin/echo "No. of patterns : $VALID"  >>heart-v1-valid.pat
/bin/echo "No. of input units : 29"  >>heart-v1-valid.pat
/bin/echo "No. of output units : 2"  >>heart-v1-valid.pat
FROM=`expr $TRAIN + 1`
tail -n +$FROM temp1.csv  | head -$VALID >temp-heart-v1-train.csv


cut -d' ' -f1,2,3,4 temp-heart-v1-train.csv |\
  sed -e "s/female/0 1/g" |\
  sed -e "s/male/1 0/g" |\
  sed -e "s/atyp_angina/0 0 0 1/g" |\
  sed -e "s/asympt/0 1 0 0/g"|\
  sed -e "s/non_anginal/0 0 1 0/g" |\
  sed -e "s/typ_angina/1 0 0 0/g" >>temp-1234.csv

cut -d' ' -f5,6,7,8,9 temp-heart-v1-train.csv |\
  sed -e "s/left_vent_hyper/1 0 0/g" |\
  sed -e "s/st_t_wave_abnormality/0 0 1/g" |\
    sed -e "s/normal/0 1 0/g"|\
  sed -e "s/t/1 0/g" |\
  sed -e "s/f/0 1/g" >>temp-56789.csv


cut -d' ' -f10,11,12 temp-heart-v1-train.csv |\
  sed -e "s/no/1 0/g" |\
  sed -e "s/yes/0 1/g" |\
  sed -e "s/down/1 0 0/g" |\
  sed -e "s/flat/0 1 0/g"|\
  sed -e "s/up/0 0 1/g" >>temp-101112.csv
cut -d' ' -f13 temp-heart-v1-train.csv |\
  sed -e "s/a/1 0 0 0/g"|\
  sed -e "s/b/0 1 0 0/g"|\
  sed -e "s/c/0 0 1 0/g"|\
  sed -e "s/d/0 0 0 1/g"|\
  sed -e "s/?/0 0 0 0/g" >>temp-13.csv

cut -d' ' -f14,15 temp-heart-v1-train.csv |\
  sed -e "s/fixed_defect/1 0 0/g" |\
  sed -e "s/normal/0 1 0/g"|\
  sed -e "s/reversable_defect/0 0 1/g" |\
  sed -e "s/?/0 0 0/g"  |\
sed -e "s/<50/1 0/g" |\
sed -e "s/>50_1/0 1/g" >>temp-1415.csv



paste -d, $FILE temp-1234.csv temp-56789.csv temp-101112.csv temp-13.csv temp-1415.csv |\
sed -e "s/,/ /g"|tr -d "\r" >> heart-v1-valid.pat
rm -r temp-1234.csv temp-56789.csv temp-101112.csv temp-13.csv temp-1415.csv temp-heart-v1-train.csv



# The test file
/bin/echo "SNNS pattern definition file V3.2"  >heart-v1-test.pat
```

```
/bin/echo "generated at Mon SEP 25 15:58:23 2018"   >>heart-v1-test.pat
/bin/echo ""  >>heart-v1-test.pat
/bin/echo ""  >>heart-v1-test.pat
/bin/echo "No. of patterns : $TEST"  >>heart-v1-test.pat
/bin/echo "No. of input units : 29"  >>heart-v1-test.pat
/bin/echo "No. of output units : 2"  >>heart-v1-test.pat
FROM=`expr $FROM + $VALID`
tail -n +$FROM temp1.csv | head -$TEST  >temp-heart-v1-train.csv

cut -d' ' -f1,2,3,4 temp-heart-v1-train.csv |\
  sed -e "s/female/0 1/g" |\
  sed -e "s/male/1 0/g" |\
  sed -e "s/atyp_angina/0 0 0 1/g" |\
  sed -e "s/asympt/0 1 0 0/g"|\
  sed -e "s/non_anginal/0 0 1 0/g" |\
  sed -e "s/typ_angina/1 0 0 0/g" >>temp-1234.csv

cut -d' ' -f5,6,7,8,9 temp-heart-v1-train.csv |\
  sed -e "s/left_vent_hyper/1 0 0/g" |\
  sed -e "s/st_t_wave_abnormality/0 0 1/g" |\
    sed -e "s/normal/0 1 0/g"|\
  sed -e "s/t/1 0/g" |\
  sed -e "s/f/0 1/g" >>temp-56789.csv


cut -d' ' -f10,11,12 temp-heart-v1-train.csv |\
  sed -e "s/no/1 0/g" |\
  sed -e "s/yes/0 1/g" |\
  sed -e "s/down/1 0 0/g" |\
  sed -e "s/flat/0 1 0/g"|\
  sed -e "s/up/0 0 1/g" >>temp-101112.csv

cut -d' ' -f13 temp-heart-v1-train.csv |\
  sed -e "s/a/1 0 0 0/g"|\
  sed -e "s/b/0 1 0 0/g"|\
  sed -e "s/c/0 0 1 0/g"|\
  sed -e "s/d/0 0 0 1/g"|\
  sed -e "s/?/0 0 0 0/g" >>temp-13.csv

cut -d' ' -f14,15 temp-heart-v1-train.csv |\
  sed -e "s/fixed_defect/1 0 0/g" |\
  sed -e "s/normal/0 1 0/g"|\
  sed -e "s/reversable_defect/0 0 1/g" |\
  sed -e "s/?/0 0 0/g"  |\
sed -e "s/<50/1 0/g" |\
sed -e "s/>50_1/0 1/g" >>temp-1415.csv



paste -d, $FILE temp-1234.csv temp-56789.csv temp-101112.csv temp-13.csv temp-1415.csv |\
sed -e "s/,/ /g" |tr -d "\r" >> heart-v1-test.pat
rm -r temp-1234.csv temp-56789.csv temp-101112.csv temp-13.csv temp-1415.csv temp-heart-v1-
train.csv
```

PART2:

```
#!/bin/bash
TRAIN=303
VALID=121
TEST=182

tail -n +20 heart-v1-pattern2.arff |\
  fgrep -v "%" | shuf |\
  sed -e"s/,/ /g"> temp2.csv

# split file merge weight attribute to the last one
cut -d' ' -f1 temp2.csv  >>temp-1.csv

cut -d' ' -f15 temp2.csv |\
  sed -e "s/<50/1 0/g" |\
  sed -e "s/>50_1/0 1/g" >>temp-15.csv

cut -d' ' -f2 temp2.csv  >>temp-2.csv

cut -d' ' -f3,4 temp2.csv |\
  sed -e "s/female/0 1/g" |\
  sed -e "s/male/1 0/g" |\
  sed -e "s/atyp_angina/0 0 0 1/g" |\
  sed -e "s/asympt/0 1 0 0/g"|\
  sed -e "s/non_anginal/0 0 1 0/g" |\
  sed -e "s/typ_angina/1 0 0 0/g"    >>temp-34.csv

cut -d' ' -f5,6,7,8,9 temp2.csv |\
  sed -e "s/left_vent_hyper/1 0 0/g" |\
  sed -e "s/st_t_wave_abnormality/0 0 1/g" |\
    sed -e "s/normal/0 1 0/g"|\
  sed -e "s/t/1 0/g" |\
  sed -e "s/f/0 1/g" >>temp-56789.csv
```

```
cut -d' ' -f10,11,12  temp2.csv |\
  sed -e "s/no/1 0/g" |\
  sed -e "s/yes/0 1/g" |\
  sed -e "s/down/1 0 0/g" |\
  sed -e "s/flat/0 1 0/g"|\
  sed -e "s/up/0 0 1/g" >>temp-101112.csv

cut -d' ' -f13  temp2.csv|\
  sed -e "s/a/1 0 0 0/g"|\
  sed -e "s/b/0 1 0 0/g"|\
  sed -e "s/c/0 0 1 0/g"|\
  sed -e "s/d/0 0 0 1/g"|\
  sed -e "s/?/0 0 0 0/g" >>temp-13.csv

cut -d' ' -f14  temp2.csv |\
  sed -e "s/fixed_defect/1 0 0/g" |\
  sed -e "s/normal/0 1 0/g"|\
  sed -e "s/reversable_defect/0 0 1/g" |\
  sed -e "s/?/0 0 0/g" >>temp-14.csv



paste -d, $FILE temp-1.csv temp-15.csv temp-34.csv temp-56789.csv temp-101112.csv temp-13.csv
temp-14.csv temp-2.csv|\
sed -e "s/,/ /g" |tr -d "\r">> temp2.pat
rm -r temp-1.csv temp-34.csv temp-56789.csv temp-101112.csv temp-13.csv temp-14.csv temp-
15.csv temp-2.csv

# The training file
/bin/echo "SNNS pattern definition file V3.2"  >heart-v2-train.pat
/bin/echo "generated at Mon SEP 25 15:58:23 2018"  >>heart-v2-train.pat
/bin/echo ""  >>heart-v2-train.pat
/bin/echo ""  >>heart-v2-train.pat
/bin/echo "No. of patterns : $TRAIN"  >>heart-v2-train.pat
/bin/echo "No. of input units : 30"  >>heart-v2-train.pat
/bin/echo "No. of output units : 1"  >>heart-v2-train.pat

head -$TRAIN temp2.pat>>heart-v2-train.pat

#cut EVERY attributes of the data

# The validation file
```

```
/bin/echo "SNNS pattern definition file V3.2"  >heart-v2-valid.pat
/bin/echo "generated at Mon SEP 25 15:58:23 2018"  >>heart-v2-valid.pat
/bin/echo ""  >>heart-v2-valid.pat
/bin/echo ""  >>heart-v2-valid.pat
/bin/echo "No. of patterns : $VALID"  >>heart-v2-valid.pat
/bin/echo "No. of input units : 30"  >>heart-v2-valid.pat
/bin/echo "No. of output units : 1"  >>heart-v2-valid.pat
FROM=`expr $TRAIN + 1`
tail -n +$FROM temp2.pat| head -$VALID >>heart-v2-valid.pat




# The test file
/bin/echo "SNNS pattern definition file V3.2"  >heart-v2-test.pat
/bin/echo "generated at Mon SEP 25 15:58:23 2018"   >>heart-v2-test.pat
/bin/echo ""  >>heart-v2-test.pat
/bin/echo ""  >>heart-v2-test.pat
/bin/echo "No. of patterns : $TEST"  >>heart-v2-test.pat
/bin/echo "No. of input units : 30"  >>heart-v2-test.pat
/bin/echo "No. of output units : 1"  >>heart-v2-test.pat
FROM=`expr $FROM + $VALID`
tail -n +$FROM temp2.pat| head -$TEST >>heart-v2-test.pat
```

Normalise Code:

```
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;

public class NormaliseCode {
	static String content = "";
	static int count = 0;

	public static void main(String[] args) {
		writeFromFile("C:/Users/user/Desktop", "heart-v1-weight.csv");
	}

	private static void writeFromFile(String absolutePath, String fileName) {
		File inFile = new File(absolutePath + "/" + fileName);
		File outFile = new File(absolutePath + "/normalizasion_" + fileName);
		try (PrintWriter writer = new PrintWriter(outFile.getAbsoluteFile(), "UTF-8")) {

			try (Scanner input = new Scanner(inFile)) {
					while (input.hasNext()) {
						content = input.next();
						if (content.equals("Weight")) {
							content = "Weight\n";
						} else {

							Double in = Double.parseDouble(content);
							if (in < 57.774) {
								double d4 = (double) (Math.round((in - 40.5792)
/ 17.19 * 10000) / 10000.0);
								double d5 = (double) (Math.round((1 - d4) *
10000) / 10000.0);

								content = d5 + "," + d4 + ",0,0,0\n";
							} else if (in >= 57.774 && in < 75.7) {
								double d4 = (double) (Math.round((in - 57.774) /
17.926 * 10000) / 10000.0);
								double d5 = (double) (Math.round((1 - d4) *
10000) / 10000.0);
								content = "0," + d5 + "," + d4 + ",0,0\n";
							} else if (in >= 75.7) {
								double d4 = (double) (Math.round((in - 75.7) /
44.1769 * 10000) / 10000.0);
								double d5 = (double) (Math.round((1 - d4) *
10000) / 10000.0);
								content = "0,0,0," + d5 + "," + d4 + "\n";
							}
						}
						count++;
						writer.write(content);
					}
				} catch (Exception e) {
					System.out.println(e.getMessage());
				}
				writer.close();
				System.out.println("File " + outFile.getName() + " has been saved to " +
absolutePath);
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}
		}
}
```

Reverse Code:

```
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;

public class ReverseCode {
        static String content = "";

        public static void main(String[] args) {
                writeFromFile("C:/Users/user/Desktop", "normalizasion_heart-v1-weight.csv");

        }

        private static void writeFromFile(String absolutePath, String fileName) {
                File inFile = new File(absolutePath + "/" + fileName);
                File outFile = new File(absolutePath + "/reversion_heart-v1-weight.csv");
                try (PrintWriter writer = new PrintWriter(outFile.getAbsoluteFile(), "UTF-8")) {

                        try (Scanner input = new Scanner(inFile)) {

                                while (input.hasNext()) {
                                        content = input.next();
                                        if (content.equals("Weight")) {
                                                content = "Weight\n";
                                        } else {
                                                System.out.println(content);
                                                Double in1 = Double.parseDouble(content.split(",")[0]);
                                                Double in2 = Double.parseDouble(content.split(",")[1]);
                                                Double in3 = Double.parseDouble(content.split(",")[2]);
                                                Double in4 = Double.parseDouble(content.split(",")[3]);
                                                Double in5 = Double.parseDouble(content.split(",")[4]);

                                                double max = findMax(in1, in2, in3, in4, in5);
                                                double d4 = 0;
                                                if (in1 == max) {
                                                        d4 = (double) (Math.round(((1 - in1) * 17.19 +
40.5792) * 10000) / 10000.0);
                                                } else if (in2 == max) {
                                                        d4 = (double) (Math.round((in2 * 17.19 +
40.5792) * 10000) / 10000.0);
                                                } else if (in3 == max) {
                                                        d4 = (double) (Math.round((in3 * 17.926 +
57.774) * 10000) / 10000.0);
```

```
                                                } else if (in4 == max) {
                                                        d4 = (double) (Math.round((in4 * 44.1769 + 75.7)
* 10000) / 10000.0);
                                                } else if (in5 == max) {
                                                         d4 = (double) (Math.round(((1 - in5) * 44.1769 +
75.7) * 10000) / 10000.0);
                                                }
                                                content = d4 + "\n";
                                        }
                                        writer.write(content);
                                }
                        } catch (Exception e) {
                                System.out.println(e.getMessage());
                        }
                        writer.close();
                        System.out.println("File " + outFile.getName() + " has been saved to " +
absolutePath);
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }
        }

        public static double findMax(double in1, double in2, double in3, double in4, double in5) {
                double max = in1;
                if (in2 > max)
                        max = in2;
                if (in3 > max)
                        max = in3;
                if (in4 > max)
                        max = in4;
                if (in5 > max)
                        max = in5;
                return max;

        }
}
```