



CSE 4088

INTRODUCTION TO MACHINE LEARNING

FINAL REPORT

Comparison of
Consistency-based & Correlation-based
Feature Selection Algorithms

1. Project Members:	<i>Zafer Emre OCAK</i>	<i>150113075</i>
	<i>Ozan GÜLHAN</i>	<i>150114013</i>

2. Abstract:

In machine learning algorithms, one of the biggest problems is scarcity of time and requirement of huge datasets. Feature selection provides optimization for these matters. Reducing time and amount of data required can change the fitness of the model significantly. By aiming this, feature selection algorithms are very important.

In order to see those effects, we consider implementing the algorithms which are “consistency-based” and “correlation-based” feature selection. In addition to that, we can also compare the required amount of time and data to reach the same fitness levels for the model.

To pick up a model, we have had a lot of different options but among them, we have decided on Bank Marketing Dataset. One of the most desired features of our project was finding an appropriate data. We have searched on the internet and it has two essential properties that first, high number of features and second, a lot of instances available. In the model, the classification goal is to predict if the client will subscribe (yes/no) a term deposit.

Overall, we would like to reduce number of attributes and to decrease least amount of time to train the model for the same fitness levels.

3. Overview of the Project

- Finding dataset and preparing it for the feature selectors.
- Constructing the ANN model.
- Training model with raw dataset which means we didn't modify dataset at all.
- Running the first selector and obtaining the best subset among all the candidate subsets.
- Running the second selector and obtaining the best subset among all the candidate subsets.
- Training the model with new subsets given by feature selectors.
- Analyzing the results.

<u>Ozan's Part</u>	<u>Emre's Part</u>
Feature selectors were available as an open source packages in the Weka library. I created a new class which calls selectors and get the results. I gave the dataset location and run the selectors. Generated subsets were given to ANN.	I searched on the internet and construct an artificial neural network in Keras library in Python. I got familiar with the concept of layers, epochs and dataset features. In order to utilize feature selection algorithms, I had to have knowledge on data dimensionality. We analyzed data together.

4. Accomplishment:

i. Finding dataset

At first, we were struggling to find a suitable dataset for our model. Because, in some cases available datasets may not be convenient for the model to be used. Thus, we have found a dataset which contains 10000 people's bank information. This dataset was an Excel file has a format of comma separated values(.csv). So, at the last column, the dataset has labels of each customer's credit approval. All the labels have binary output indicating 1s and 0s.

ii. Building ANN model

To begin with, we have made a comprehensive research on the internet to figure out the Keras library and its functions. As constructing the model we put 1 input-layer, 2 hidden-layer which have activation function of ReLu and 1 output-layer for generating the classification output. Our input layer receives 11 features as inputs because out of 13 features, one of them is for the labelling data and one of them was unnecessary feature - *row number* - which doesn't any effect on the output result.

iii. Model training on raw data

Training with raw data, the model had 11 dimensions as inputs and 1 binary output. Before modification of the dataset via feature selection, these were the total number of features in the model. We tracked the total duration via putting start and end temporary variables and in addition to that, we have analyzed the accuracy rate after the last epoch the model has passed.

iv. Running the Selectors

a. Correlation-based Feature Selection

- ‘Good feature subsets contain features highly correlated with the classification , yet uncorrelated to each other.’ [2]
- This means that, the more a feature is uncorrelated with other features in the subset, it has a good effect on the merit of subset, also the more it's correlated with the class , it also has a good effect.

$$\text{Merit}_{S_k} = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}$$

✚ The following equation gives the merit of a feature subset S consisting of k features.

✚ R_{cf} is the average value of all feature-classification correlations.

✚ R_{ff} is the average value of all feature-feature correlations.

b. Consistency-based Feature Selection

- When this algorithm takes the candidate subset, it looks for all the instances where their attribute values match but their class labels are different. This kind of patterns are inconsistent. Count of these patterns is the inconsistency count.
- Inconsistency rate of a subset S is the sum of all the inconsistency counts over all patterns of the feature subset that appears in the data divided by total number of instances.
- The less this number for a subset, the better for that subset.

v. Training the model with new subsets

This part is the main concept of our project. We have given the substituted features from the selector algorithms and gave the model to observe the differences.

So we have a graph below that clearly explaining what is happening in different datasets with different selection methods.

	Total Features Used	Accuracy Rate	Total Duration
Raw Model	11	%83.60	83 seconds
Correlation-based	8	%85.61	85 seconds
Consistency-based	7	%83.41	80 seconds

vi. Analyzing the results and experiments carried out

At this part, we illustrate the experiment results as below:

```

17 labelencoder_X_1 = LabelEncoder()
18 X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
19 labelencoder_X_2 = LabelEncoder()
20 X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
21 onehotencoder = OneHotEncoder(categorical_features = [1])
22 X = onehotencoder.fit_transform(X).toarray()
23 X = X[:, 1:]
24
25 #Splitting the dataset into the Training set and Test set
26 from sklearn.model_selection import train_test_split
27 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
28
29 #Feature Scaling
30 from sklearn.preprocessing import StandardScaler
31 sc = StandardScaler()
32 X_train = sc.fit_transform(X_train)
33 X_test = sc.transform(X_test)
34
35 #PART 2 - Making the ANN Model
36
37 #Importing the Keras Libraries and packages
38 import keras
39 from keras.models import Sequential
40 from keras.layers import Dense
41
42
43 #Initializing the ANN
44 classifier = Sequential()
45
46 #Adding the Input Layer and the first hidden layer
47 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 11))
48 #classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 8))
49 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 7))
50 #Adding the second hidden layer
51 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
52 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
53 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
54 #Adding the output layer
55 classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
56
57 #Compiling the ANN - adding stochastic gradient descent
58 classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
59
60 start = datetime.datetime.now()
61
62 #fitting the ANN to the training set
63 classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
64
65 end = datetime.datetime.now()
66 #Calculate the total duration of training
67 total_training_time = end - start
68

```

```

Epoch 91/100
8000/8000 [=====] - 1s 99us/step - loss: 0.3942 - acc: 0.8359
Epoch 92/100
8000/8000 [=====] - 1s 101us/step - loss: 0.3938 - acc: 0.8348
Epoch 93/100
8000/8000 [=====] - 1s 111us/step - loss: 0.3943 - acc: 0.8356
Epoch 94/100
8000/8000 [=====] - 1s 101us/step - loss: 0.3939 - acc: 0.8368
Epoch 95/100
8000/8000 [=====] - 1s 108us/step - loss: 0.3937 - acc: 0.8380
Epoch 96/100
8000/8000 [=====] - 1s 101us/step - loss: 0.3939 - acc: 0.8368
Epoch 97/100
8000/8000 [=====] - 1s 108us/step - loss: 0.3939 - acc: 0.8351
Epoch 98/100
8000/8000 [=====] - 1s 101us/step - loss: 0.3943 - acc: 0.8351
Epoch 99/100
8000/8000 [=====] - 1s 101us/step - loss: 0.3940 - acc: 0.8358
Epoch 100/100
8000/8000 [=====] - 1s 101us/step - loss: 0.3942 - acc: 0.8365
Total training time: 0:01:22.110553

```

These experiment shows the model accuracy and total training time with *raw dataset*.

```

42
43 #Initializing the ANN
44 classifier = Sequential()
45
46 #Adding the Input Layer and the first hidden layer
47 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 11))
48 #classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 8))
49 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 7))
50 #Adding the second hidden layer
51 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
52 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
53 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
54 #Adding the output layer
55 classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
56
57 #Compiling the ANN - adding stochastic gradient descent
58 classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
59
60 start = datetime.datetime.now()
61
62 #fitting the ANN to the training set
63 classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
64
65 end = datetime.datetime.now()
66 #Calculate the total duration of training
67 total_training_time = end - start
68

```

```

Epoch 95/100
8000/8000 [=====] - 1s 107us/step - loss: 0.3507 - acc: 0.8551
Epoch 96/100
8000/8000 [=====] - 1s 105us/step - loss: 0.3518 - acc: 0.8572
Epoch 97/100
8000/8000 [=====] - 1s 105us/step - loss: 0.3531 - acc: 0.8557
Epoch 98/100
8000/8000 [=====] - 1s 106us/step - loss: 0.3520 - acc: 0.8556
Epoch 99/100
8000/8000 [=====] - 1s 106us/step - loss: 0.3519 - acc: 0.8570
Epoch 100/100
8000/8000 [=====] - 1s 106us/step - loss: 0.3520 - acc: 0.8561
Total training time: 0:01:25.883130

```

These experiment shows the model accuracy and total training time with *the dataset correlation-based selector has been applied*.

The screenshot shows a Jupyter Notebook with two parts. On the left, the code for training a neural network is displayed, including adding an output layer, compiling the model with 'adam' optimizer and 'binary_crossentropy' loss, fitting it to the training set, and calculating the total training time. On the right, the console output shows the progress of the training, including epoch numbers, loss, and accuracy. The final output shows a total training time of 0:01:20.807299 and an accuracy of 0.8341.

```
53 #adding the output layer
54 classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
55 #compiling the ANN - adding stochastic gradient descent
56 classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
57 start = datetime.datetime.now()
58 #Fitting the ANN to the training set
59 classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
60 #Calculate the total duration of training
61 end = datetime.datetime.now()
62 print("Total training time: ", end-start)
63 #Predicting the Test set results
64 y_pred = classifier.predict(X_test)
65 y_pred = (y_pred > 0.5)
66 #Making the Confusion Matrix
67 from sklearn.metrics import confusion_matrix
68 cm = confusion_matrix(y_test, y_pred)
69
```

Console I/A

```
Epoch 91/100 [=====] - 1s 100us/step - loss: 0.4025 - acc: 0.8342
Epoch 92/100 [=====] - 1s 100us/step - loss: 0.4022 - acc: 0.8327
Epoch 93/100 [=====] - 1s 101us/step - loss: 0.4026 - acc: 0.8334
Epoch 94/100 [=====] - 1s 98us/step - loss: 0.4023 - acc: 0.8344
Epoch 95/100 [=====] - 1s 100us/step - loss: 0.4025 - acc: 0.8342
Epoch 96/100 [=====] - 1s 100us/step - loss: 0.4025 - acc: 0.8345
Epoch 97/100 [=====] - 1s 100us/step - loss: 0.4020 - acc: 0.8337
Epoch 98/100 [=====] - 1s 100us/step - loss: 0.4023 - acc: 0.8329
Epoch 99/100 [=====] - 1s 98us/step - loss: 0.4017 - acc: 0.8346
Epoch 100/100 [=====] - 1s 100us/step - loss: 0.4024 - acc: 0.8341
Total training time: 0:01:20.807299
```

These experiment shows the model accuracy and total training time with *the dataset consistency-based selector has been applied.*

a. Installing essential libraries

In this very first step of the project, we determined we need to certify the essential libraries to be used in our model.

Imported libraries is going to be explained briefly. The libraries were:

- *Numpy Lib of Python*

Basically, this library serves as a mathematical optimizer in Python. We had an idea that in machine learning algorithms, capability of arithmetic operations is a must. Thus, since we already implement Artificial Neural Networks(ANN) in our model, we knew we would need that library beforehand.

- *Pandas Lib of Python*

Our both instance and complete dataset is in .csv format so we need to use this library to easily read the excel file.

- *LabelEncoder & OneHotEncoder Libs of Python*

These libraries serves as fit the model dataset suitable for training. Via importing them, we can change the dataset format to array format. Prior to, as soon as we read our dataset from the .csv file, it was in Object format.

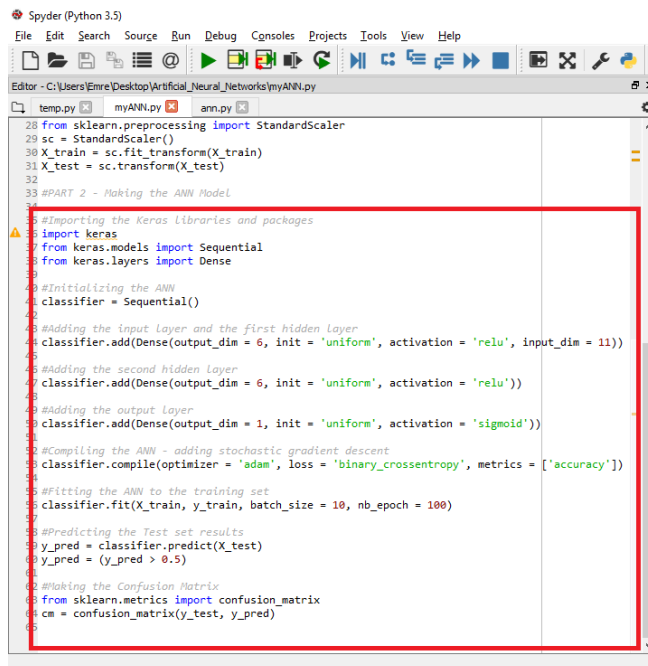
c. Constructing the neural network

As Keras library has been already mentioned, it gave us the opportunity to easily construct layers and splitting up the training and testing dataset.

```
23 #Splitting the dataset into the Training set and Test set
24 from sklearn.model_selection import train_test_split
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
26
27 #Feature Scaling
28 from sklearn.preprocessing import StandardScaler
29 sc = StandardScaler()
30 X_train = sc.fit_transform(X_train)
31 X_test = sc.transform(X_test)
32
```

In first two lines, 20% of the sample dataset is dedicated to testing part and other 80% of the sample dataset has been given to training part.

Under the label of feature scaling, X_train array has been used to altering weights and getting model to become ready for testing. X_test is obviously assigned to testing process.

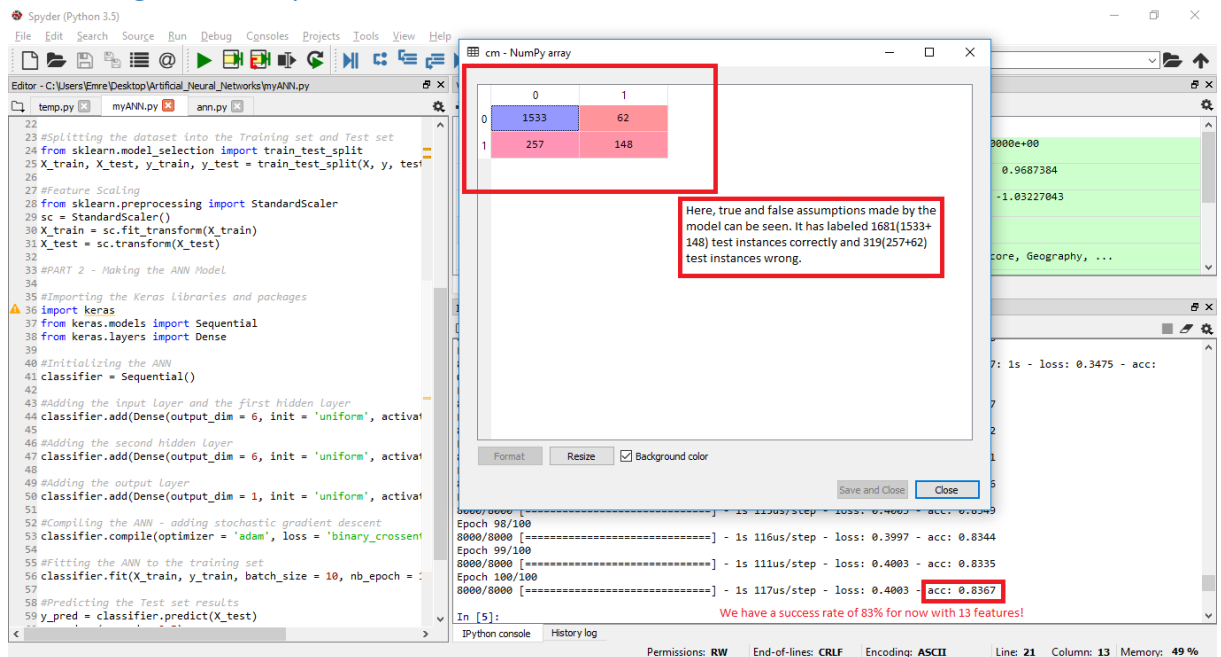


```
28 from sklearn.preprocessing import StandardScaler
29 sc = StandardScaler()
30 X_train = sc.fit_transform(X_train)
31 X_test = sc.transform(X_test)
32
33 #PART 2 - Making the ANN Model
34
35 #Importing the Keras libraries and packages
36 import keras
37 from keras.models import Sequential
38 from keras.layers import Dense
39
40 #Initializing the ANN
41 classifier = Sequential()
42
43 #Adding the input layer and the first hidden layer
44 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 11))
45
46 #Adding the second hidden layer
47 classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
48
49 #Adding the output layer
50 classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
51
52 #Compiling the ANN - adding stochastic gradient descent
53 classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
54
55 #Fitting the ANN to the training set
56 classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
57
58 #Predicting the Test set results
59 y_pred = classifier.predict(X_test)
60 y_pred = (y_pred > 0.5)
61
62 #Making the Confusion Matrix
63 from sklearn.metrics import confusion_matrix
64 cm = confusion_matrix(y_test, y_pred)
65
```

This part of our program code, priorly, the corresponding import process has been done and afterwards, we determined the layers and the particular weight adjustment methods have been determined which are relu and sigmoid functions. In this model, we have 3 layers and 13 total of neurons. The weight's initial values are set according to uniform distribution.

With a batch size of 10 which means the data is going to be grouped by 10 for each iteration and will be given. Also, number of epoch is 100 so this means that our model will be trained by that dataset for 100 times.

d. Observing the initial performance of the model



We have 2 different possible outcomes of predictions, 1 or 0... The total correct and false answers can be shown with an explanation. Our current success rate is 83% with 13 different features.

5. Summary:

To briefly talking about the findings in this project, we have got a general understanding of feature selection algorithms and their corresponding reports. Moreover, we got familiar with the existing machine learning technologies and use them in order to develop a model. There were actually quite different concepts in that area. For instance, if a data scientist would like to process an image dataset then he/she probably need to handle convolutional neural networks.

Furthermore, we had significant amount of experience about the dataset compatibility for the ANN models. For example, we have dealt with data types for using consistency-based algorithm because the dataset compatibility with selector is essential.

We also altered the number of layers and the number of epochs in our model to observe the improvement in the model.

All these actions greatly contributed our knowledge of machine learning concept. There were ideas and undiscovered methods we could use in this project. To simply mention, we consider utilizing cross-validation and picking different model selection methods in collaboration with different model structures and with different classification problems.

6. References:

- [1] Manoranjan Dash, Huan Liu, Consistency-based search in feature selection, Artificial Intelligence 151 (2003) 155–176
- [2] Mark A. Hall, Correlation-based Feature Selection for Machine Learning, 1999
- [3] <https://keras.io/getting-started/sequential-model-guide/>