Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2016-2017

Adrian Groza, Anca Marginean and Radu Razvan Slavescu
Tool: Belief and Decision Networks - Version 5.1.10

Name:Brincoveanu Vasile Vlad si Ghiurca Dorin
Group:30233
Email:gg.vladbrincoveanu@gmail.com

Assoc. Prof. dr. eng. Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

Installing the tool (W_1)

The teaching objectives for this week are:

1. To know how to use the tool in our future project.
2. To know from where we can download it, what version, etc. And how to run it.

Steps for installing the tool on windows or mac:

1. First step is to go on the application site
2. Then you need to go on the installation part of the site.
3. Under installing the latest release section it can be found the linux commands for installing scikit-learn prerequisites.

- Some notes: you should have python (2.7-eq. or greater, or 3.3 - eq. or greater), NumPy (1.8.2 - eq. or greater), SciPy(0.13.3 - eq. or greater) installed prior to running the application, otherwise it won't work.

Chapter 2

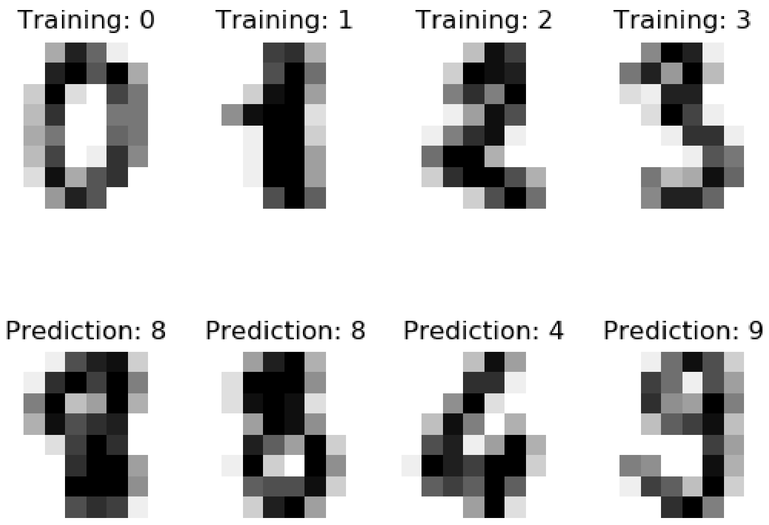
Running and understanding examples (W_2)

The teaching objectives for this week are:

1. To run and understand the some examples.
2. To identify what realistic problems are adequate for our tool.

- First example

  We can use scikit-learn to recognize images of hand-written digits.



```
"""
================================
Recognizing hand-written digits
================================

An example showing how the scikit-learn can be used to recognize
```

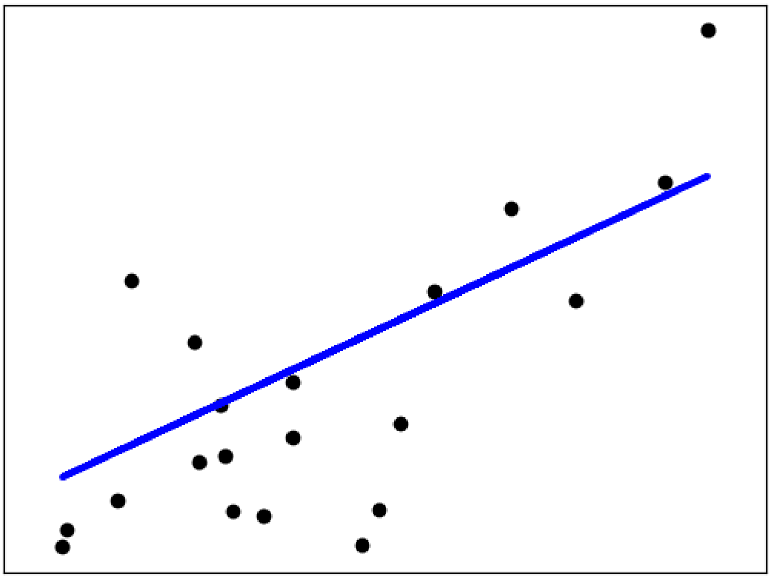
```
images of hand-written digits.

This example is commented in the
:ref:`tutorial section of the user manual <introduction>`.

"""
print(__doc__)

# Author: Gael Varoquaux <gael dot varoquaux at normalesup dot
org>
# License: BSD 3 clause

# Standard scientific Python imports
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics

# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of
#digits, let's  have a look at the first 4 images, stored in the
#`images` attribute of the dataset.  If we were working from
# image files, we could load them using matplotlib.pyplot.imread.
#Note that each image must have the same size. For these
# images, we know which digit they represent: it is given in the
#'target' of the dataset.
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image,cmap=plt.cm.gray_r,interpolation='nearest')
    plt.title('Training: %i' % label)

# To apply a classifier on this data, we need to flatten the
#image, to turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
classifier = svm.SVC(gamma=0.001)

# We learn the digits on the first half of the digits
classifier.fit(data[:n_samples //2],digits.target[:n_samples //2])

# Now predict the value of the digit on the second half:
expected = digits.target[n_samples // 2:]
predicted = classifier.predict(data[n_samples // 2:])
```

```
print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predic

images_and_predictions = list(zip(digits.images[n_samples // 2:], predict
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)

plt.show()
```

- Second example - Linear Regression

  This example uses the only the first feature of the diabetes dataset, in order to illustrate a two-dimensional plot of this regression technique. The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

  The coefficients, the residual sum of squares and the variance score are also calculated.



```
print(__doc__)


# Code source: Jaques Grobler
# License: BSD 3 clause
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()


# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred)

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

Chapter 3

Understanding conceptual instrumentation (W_3)

The teaching objectives for this week are:

1. To understand the algorithm(s) on which your tool relies.
2. To get used with writing algorithms in Latex

- Explaining the tool

  Machine learning algorithms implemented in scikit-learn expect data to be stored in a two-dimensional array or matrix. The arrays can be either numpy arrays, or in some cases scipy.sparse matrices. The size of the array is expected to be: [n samples, n features].

  scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the boston house prices dataset for regression. A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the .data member, which is a n samples, n features array. In the case of supervised problem, one or more response variables are stored in the .target member. More details on the different datasets can be found in the dedicated section.

  In the case of the digits dataset, the task is to predict, given an image, which digit it represents. We are given samples of each of the 10 possible classes (the digits zero through nine) on which we fit an estimator to be able to predict the classes to which unseen samples belong.

  In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).

Chapter 4

Project description (W_4)

The teaching objectives for this week are:

1. To have a clear description of what you intend to develop.
2. To point to specific resources (datasets, knowledge bases, external tools) that support the development of your idea and which minimise the risk of failure.
3. To identify related work (articles) that are relevant or similar to your approach.

4.1 Describe in natural language

We propose to predict the chance of having a heart disease or having hypertension given the dataset.

Our approach was to use neural networks and to try different algorithms integrated in Multi-layer Perceptron (MLP) and compare which one of those works better for the given dataset of heart disease. More detailed info will be described in implementation chapter.

We will start by examinin our dataset .[1] . The datasets are provided by SaumyaAgarwal HealthCare Problem: Prediction Stroke Patients on the kaggle website. He used it to predict stroke by having different data. We will use it for predicting heart disease or having hypertension.

Understanding Data Here is the Definitions of the columns of the data
id-Patient ID
gender-Gender of Patient
age-Age of Patient
hypertension-0 - no hypertension, 1 - suffering from hypertension
heart-disease-0 - no heart disease, 1 - suffering from heart disease
ever-married-Yes/No
work-type-Type of occupation
Residence-type-Area type of residence (Urban/ Rural)
avg-glucose-level-Average Glucose level (measured after meal)
bmi-Body mass index
smoking-status-patient's smoking status

[1]https://www.kaggle.com/asaumya/healthcare-problem-prediction-stroke-patients

stroke-0 - no stroke, 1 - suffered stroke

The columns that we will be using are all except id-patient and stroke, to not interfere with what our poster used the data for.

Furthermore, there are 2 datasets, one for training and one for testing. We merged all , and use cross-validation function to split the data into test data and train data.

All this data will be converted to integers, usign different functions of the panda library.

Chapter 5

Algorithms (W_8)

- Neural network models

  Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function f(dot): R to the power m goest to R to the power o by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features X = x1, x2, ..., xm and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 1 shows a one hidden layer MLP with scalar output.

  The advantages of Multi-layer Perceptron are:

  - capability to learn non-linear models.

  - capability to learn models in real-time (on-line learning) using partial fit.

  The disadvantages of Multi-layer Perceptron (MLP) include:

  - MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.

  - MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.

  - MLP is sensitive to feature scaling.

- Classifier

  Class MLPClassifier implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

  MLP trains on two arrays: array X of size (n samples, n features), which holds the training samples represented as floating point feature vectors; and array y of size (n samples,), which holds the target values (class labels) for the training samples.

  Currently, MLPClassifier supports only the Cross-Entropy loss function, which allows probability estimates by running the predict proba method.

  MLP trains using Backpropagation. More precisely, it trains using some form of gradient descent and the gradients are calculated using Backpropagation. For classification, it minimizes the Cross-Entropy loss function, giving a vector of probability estimates P(y—x) per sample x.

MLPClassifier supports multi-class classification by applying Softmax as the output function.

Further, the model supports multi-label classification in which a sample can belong to more than one class. For each class, the raw output passes through the logistic function. Values larger or equal to 0.5 are rounded to 1, otherwise to 0. For a predicted output of a sample, the indices where the value is 1 represents the assigned classes of that sample.

Chapter 6

Implementation details (W_5)

The teaching objectives for this week are:

1. Illustrate each aspect of the reality that you have modelled in your solution.
2. To explain the relevant code from your scenario.

6.1 Code description

We will use the pandas library to do operations on the dataset. here

```
import pandas as pd
import numpy as np

input_file = "/home/vlad/Desktop/Heart-Disease-Prediction-
using-Machine-Leaning-master/test_2v.csv"

data = pd.read_csv(input_file, header = 0)
data = data.dropna()
```

We will read all the dataset, and tell panda that the first line is where the labels are. Next we will use dropna() function to get rid of the rows in the dataset, where the data is corrupted. Datasets are very large so it is very probable that some values are corrupted or not existent.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data_1 = data.apply(le.fit_transform)
```

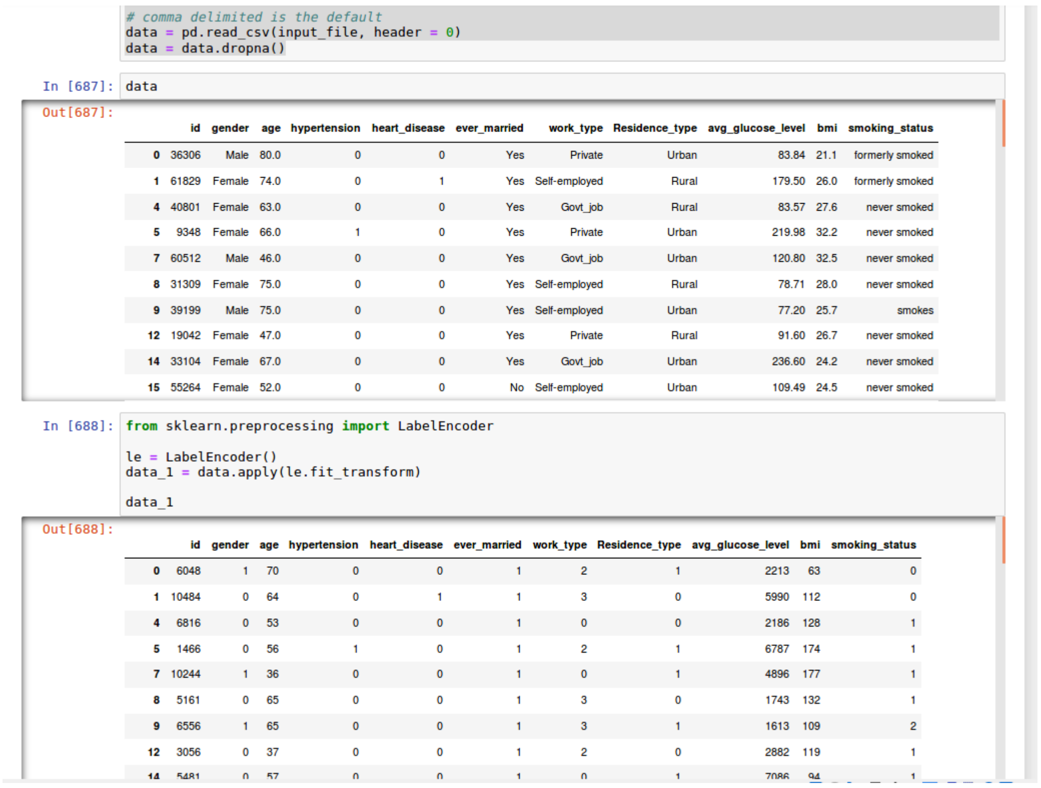
Next we will encode the text in the database. We don't have rows only with integer values, so the LabelEncoder will get thru every column and convert text and float to integer.

More, LabelEncoder will encode the text like this:

In the coloumn gender, for example, we have Male and Female values, the encoder will replace Male with = 0, and Female with 1.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data_1 = data.apply(le.fit_transform)
```

This is the difference of Encoded data vs Normal dataset



```
# comma delimited is the default
data = pd.read_csv(input_file, header = 0)
data = data.dropna()
```

```
In [687]: data
```

```
Out[687]:
       id  gender   age  hypertension  heart_disease  ever_married      work_type  Residence_type  avg_glucose_level   bmi   smoking_status
 0  36306    Male  80.0             0              0           Yes        Private           Urban              83.84  21.1  formerly smoked
 1  61829  Female  74.0             0              1           Yes  Self-employed           Rural             179.50  26.0  formerly smoked
 4  40801  Female  63.0             0              0           Yes       Govt_job           Rural              83.57  27.6     never smoked
 5   9348  Female  66.0             1              0           Yes        Private           Urban             219.98  32.2     never smoked
 7  60512    Male  46.0             0              0           Yes       Govt_job           Urban             120.80  32.5     never smoked
 8  31309  Female  75.0             0              0           Yes  Self-employed           Rural              78.71  28.0     never smoked
 9  39199    Male  75.0             0              0           Yes  Self-employed           Urban              77.20  25.7           smokes
12  19042  Female  47.0             0              0           Yes        Private           Rural              91.60  26.7     never smoked
14  33104  Female  67.0             0              0           Yes       Govt_job           Urban             236.60  24.2     never smoked
15  55264  Female  52.0             0              0            No  Self-employed           Urban             109.49  24.5     never smoked
```

```
In [688]: from sklearn.preprocessing import LabelEncoder

          le = LabelEncoder()
          data_1 = data.apply(le.fit_transform)

          data_1
```

```
Out[688]:
       id  gender  age  hypertension  heart_disease  ever_married  work_type  Residence_type  avg_glucose_level  bmi  smoking_status
 0   6048       1   70             0              0             1          2               1               2213   63               0
 1  10484       0   64             0              1             1          3               0               5990  112               0
 4   6816       0   53             0              0             1          0               0               2186  128               1
 5   1466       0   56             1              0             1          2               1               6787  174               1
 7  10244       1   36             0              0             1          0               1               4896  177               1
 8   5161       0   65             0              0             1          3               0               1743  132               1
 9   6556       1   65             0              0             1          3               1               1613  109               2
12   3056       0   37             0              0             1          2               0               2882  119               1
14   5481       0   57             0              0             1          0               1               7086   94               1
```

```
data_2 = data_1.loc[data_1['hypertension'] == 1] #1390
data_3 = data_1[:2250];

result = pd.concat([data_2,data_3]);
```

Here we will maipulated the dataset, by taking n amount of rows, and trying to get different precentage of true tested data.

We can see that in our dataset there are 1390 true hypertension values, so by adding 2250 more rows, we will get 1660 true values out of 3640 which is 45 per cent true value, etc

```
x = result.drop(result.columns[[0,3]],axis=1)
y = result['hypertension']
```

Further into code, we will split the data into 2 parts. x will get all the columns that will help to make the prediction, and y will get the column that needs to be predicted. In this case hypertension. But we will go also with heart-disease, and thus we will have 2 examples.

```
from sklearn import cross_validation
X_train, X_test, y_train, y_test = cross_validation.train_test_split(x,y,test_size=0.3,r
```

Here, we split the dataset into train data and test data. The test-size will get the precentage of the dataset to be tested, and the rest will be split to train(0.7). Random state will tell how random will the ddata be splitted.

```
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC
clf = MLPClassifier(activation='identity',solver='lbfgs', alpha=1e-4, random_state=1, ve
#clf = LinearSVC(C=0.0001)
clf.fit(X_train, y_train)
y_predict=clf.predict(X_test)
```

Here we use neural network with identity activation ,no-op activation, useful to implement linear bottleneck, returns f(x) = x, and lbfgs ,is an optimizer in the family of quasi-Newton methods.And we fit the training data.

```
import numpy as np
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_predict)

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print classification_report(y_test,y_predict,target_names = ['falsep','truep'])
tn, fp, fn, tp = confusion_matrix(y_test, y_predict).ravel()
(tn, fp, fn, tp)
confusion_matrix(y_test, y_predict)
```

Finally, we will have the metrics. Acurracy score is nothing more than the average recall , whereas report we will get us the precision recall and f1-score with suport and make the average aswell. The confusion matrix will tell us how the neural network gussed the test data.

Chapter 7

Experiments and tables (W_6)

The objectives for this week are:

1. To describe and interpret each experiment that you have performed

An experiment investigates how some variables are related. Usually, experiments verify a previosly formulated hypothesis. Such hypothesis may investigate how your software degrades its performance with larger inputs. You will need to run simulations to see how your implementation is affected by different inputs.

7.1 Result interpretation and data

Here we will put all the results regarding hypertension and heart-disease from our dataset. We tested it with neural networks, with different algs and with lienarSCV and SCV ot see the diferences.

The dataset percentage refeers to the amount of true test data in the dataset. If we let all the rows of the dataset, we will see that every classifier will predict false negative, and false positive, because there is few people with hypertension.

We will manipulate the dataset in our favor, to be more precise in thesting the algorithms.

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances.Wiki.

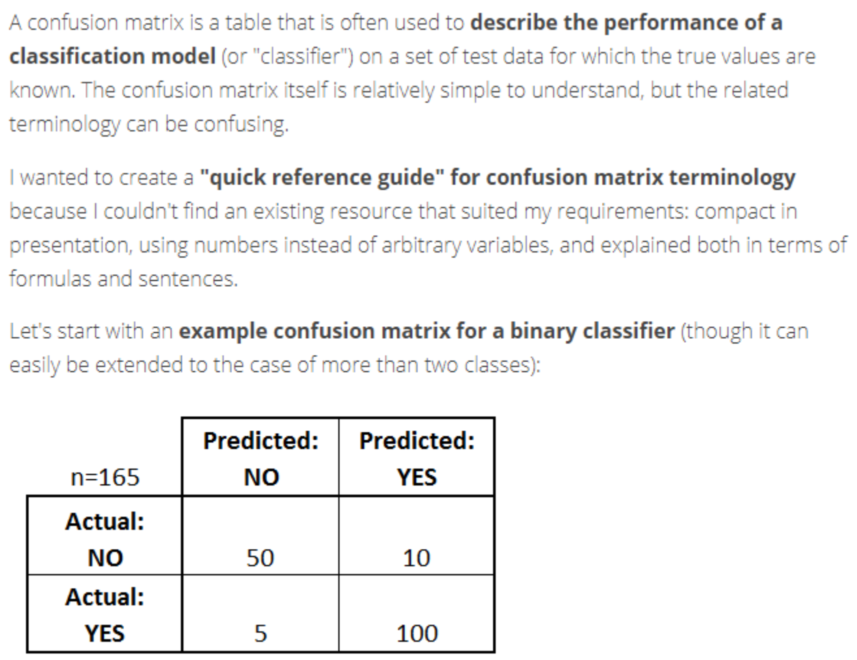
The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

'lbfgs' is an optimizer in the family of quasi-Newton methods.

'sgd' refers to stochastic gradient descent.

adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Confusion matrix



A confusion matrix is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

I wanted to create a **"quick reference guide" for confusion matrix terminology** because I couldn't find an existing resource that suited my requirements: compact in presentation, using numbers instead of arbitrary variables, and explained both in terms of formulas and sentences.

Let's start with an **example confusion matrix for a binary classifier** (though it can easily be extended to the case of more than two classes):

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Table 7.1: Results for predicting hypertension

Method and algorithm	**Dataset percentage(true)**	**Precision**	**Recall**	**F1-score**
Neural Net - lbfgs	45.6	0.69	0.69	0.69
Neural Net - lbfgs	54.11	0.58	0.54	0.39
Neural Net - sgd	45.6	0.53	0.52	0.38
Neural Net - sgd	54.11	0.50	0.54	0.39
Neural Net - adam	45.6	0.57	0.53	0.37
Neural Net - adam	54.11	0.58	0.50	0.42
All data NN-adam	0.09	0.82	0.89	0.83

```
 Where confusion matrix of 1.\\
\[414, 157\],\\
\[183, 338\]\\

\tab confusion matrix of 2.\\
\[ 7, 395\],\\
\[ 4, 461\]\\

\tab confusion of 3.
\[558, 13\],\\
\[506, 15\]\\

\tab confusion of 4.
\[ 6, 396\],\\
\[ 7, 458\]\\

\tab conf of 5.
\[566, 5\],\\
\[513, 8\]\\
\tab conf of 6.
\[368, 34\]\\
\[397, 68\]\\
```

Here we see that there is a kind of good prediction , regarding the small sample of the dataset. Ofc, if we were to put all 50k rows in we will get 95 per cent prediction but only because the neural network have been trained to predict all values false because of dataset having few 5-10 per cent values of true in the test data.

Table 7.2: Results for predicting heart-disease

Method and algorithm	**Dataset percentage(true)**	**Precision**	**Recall**	**F1-score**
Neural Net - lbfgs	45.6	0.41	0.64	0.50
Neural Net - lbfgs	54.11	0.53	0.73	0.61
Neural Net - sgd	45.6	0.63	0.64	0.51
Neural Net - sgd	54.11	0.62	0.73	0.61
Neural Net - adam	45.6	0.66	0.44	0.37
Neural Net - adam	54.11	0.67	0.73	0.64
All data NN-adam	0.09	0.91	0.95	0.93

```
Where confusion matrix of 1.\\
\[411, 0\],\\
\[233, 0\]\\

\tab confusion matrix of 2.\\
\[632, 0\],\\
\[237, 0\]\\

\tab confusion of 3.
\[409, 2\],\\
\[230, 3\]\\

\tab confusion of 4.
\[630, 2\],\\
\[236, 1\]\\

\tab conf of 5.
\[ 67, 344\],\\
\[ 16, 217\]\\

\tab conf of 6.
\[617, 15\],\\
\[223, 14\]\\

\tab conf of 7.
\[3554, 1\],\\
\[ 172, 0\]\\
```

7.2 Conclusions

The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.Skit.

Here we see that there is a kind of good prediction , regarding the small sample of the dataset. Ofc, if we were to put all 50k rows in we will get 95 per cent prediction but only because the neural network have been trained to predict all values false positive because of dataset having few 5-10 per cent values of true in the test data.

One fact, if we add hidden layers of neurons, the values will start to converge rapidily to 80+ prediction and go into false positive values, like we said in the upper paragraph.

Chapter 8

Related Work

- Approach

  Our approach was to try different classifier and try and compare which one of those works better for the given dataset of heart disease. The classifiers that we have compared are Linear SVM, Non-linear SVM and Stratified K-Mean on the given vector representation of Cleveland dataset. For our experimental purposes, we have divided our experiment into 2 problems. For both problems, we try to run our classifiers for 60/40 and 80/20 splits where we use 60 per cent and 80 per cent for training our classifiers and 40 per cent and 20 percent for testing their predictions respectively. Cleveland dataset has 303 instances and 14 attributes. Our first step is to apply dimensionality reduction and for that purpose we have use Principal Component Analysis (PCA) with 5 components. Once the PCA has been applied on the original X value where X is the feature set, our feature set is reduced to X-new which is a vector representation of 303 samples x 5 features. First, we try out 60/40 split of X-new where 60 per cent of X-new is used to train the SVM classifier and 40 per cent of X-new is used to test. Similarly, next we try 80/20 split of X-new, which is problem 1 of our experiment.

- Problem 1

  We classify data using a Linear SVM and predict likelihood of disease belonging to a particular class of severity ranging from 1 to 4 i.e. least to most severe with value of C=0.001. Here, C is the penalty that the classifier incurs every time there is a misclassification that takes place so job of the classifier is to incur penalty as minimal as possible while classifying data in order to keep cost of classification at minimum. In order to check if other type of classifiers work better for this dataset than Linear SVM, we use a RBF i.e. non-linear kernel for the SVM classifier and classify data keeping value of C same. Similarly as last part of our problem 1, we use Stratified k-fold cross validation with 5 folds with a RBF kernel and keeping value of C same as for above classifiers in our search to find which classifier works better for this dataset. The results for this have been shown below in Fig 1 below.

- Problem 2

  For problem-2 of our experiment, we go a step further by predicting absence (zero) or presence (non-zero) of heart disease. This is possible because we group all severity classes (1 to 4) together which mean that a non-zero would indicate presence of heart disease and a zero would indicate absence of heart disease. Problem-2 of the experiment follows same procedure as that of problem-1. First step is dimensionality reduction for which we use PCA with 5 components that picks best 5 components out of 14 attributes. Now what we get is a vector representation as we obtained in problem-1, which basically implies 303 samples x 5 features. For problem-2, we use an 80/20 split where 80 per cent of data is used to train classifier and 20 per cent is used to test. Now, we follow the same procedure as we did for problem-1 we apply 3 classifiers i.e. Linear SVM, Non-Linear SVM with RBF kernel and Stratified k-means cross validation with 5 folds, all for a value of C=0.001. The results are shown in fig.2.

- Results

Table 1: Problem-1 60/40 Data Split

Classifier	**Classification Accuracy (X_new)**	**Classification Accuracy (X_new- Split)**
Linear SVM	80%	55%
Non-Linear SVM (kernel 'RBF')	100%	49%
Stratified k-mean cross validation (kernel 'RBF')	100%	55%

Table 2: Problem-1 80/20 Data Split

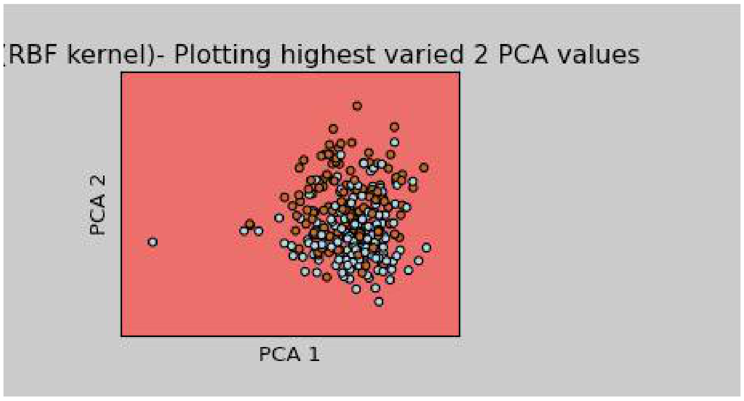
Classifier	**Classification Accuracy (X_new)**	**Classification Accuracy (X_new- Split)**
Linear SVM	81%	59%
Non-Linear SVM (kernel 'RBF')	100%	57.37%
Stratified k-mean cross validation (kernel 'RBF')	100%	54.23%

Table 3: Problem-2 80/20 Data Split

Classifier	**Classification Accuracy (X_new)**	**Classification Accuracy (X_new- Split)**
Linear SVM	100%	73.77%
Non-Linear SVM (kernel 'RBF')	100%	57.37%
Stratified k-mean cross validation (kernel 'RBF')	100%	54.23%

Figure 1: Problem-1 PCA Dimensionality Reduction (n_components=5)

Figure 2: Problem-2 Integrating all severity classes into one i.e. presence (non-zero)



- Conclusion

  Based on the results shown above and experiments performed, it is evident that input data plays an important role in prediction along with machine learning techniques. As is seen in the dataset, provided, we have labels from 0 to 4 where the labels of 4 are hardly 13 and when we split the data into train and test, the number become very less which is nothing but noise and can be totally removed from the dataset by using filtering techniques and hence the linear model will be available to predict the outcome much better with absence of noise. Moreover, PCA has again proven that we can get rid of similar feature set and still obtain predictions with great efficiency. Moreover, we have conducted tests using non linear RBF kernel which is a normal first choice and then validating against linear SVC kernel which outperformed RBF in split case. Most importantly, the above experiment not only helped us in predicting the outcome but also gave us valuable insights about the nature of data, which can be used in future to train our classifiers in a much better way.

  All source file can be founded here.

Appendix A

Your original code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
input_file = "/home/vlad/Desktop/Heart-Disease-Prediction-using-Machine-Leaning-master/t


# comma delimited is the default
data = pd.read_csv(input_file, header = 0)
data = data.dropna()

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data_1 = data.apply(le.fit_transform)
data_2 = data_1.loc[data_1['hypertension'] == 1] #1390
#x = data_1[data.columns.drop([[0,3,4]])]
#x = data_1.drop(data_1.columns[[0,3]],axis=1)
#y = data_1['hypertension']
data_3 = data_1[:2800];

result = pd.concat([data_2,data_3]);
#x = data_1[data.columns.drop([[0,3,4]])]
x = result.drop(result.columns[[0,3]],axis=1)
y = result['hypertension']

from sklearn import cross_validation
X_train, X_test, y_train, y_test = cross_validation.train_test_split(x,
y,test_size=0.4,random_state=42)

from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC
#clf = MLPClassifier(activation='logistic',solver='adam', alpha=1e-5, hidden_layer_sizes
clf = LinearSVC(C=0.0000001)
clf.fit(X_train, y_train)

print clf.score(X_test,y_test)
y_predict=clf.predict(X_test)


# np.random.seed(0)
# for _ in range(6):
# this_X = .1*np.random.normal(size=(2, 1)) + X_test
# regr.fit(this_X, y_train)
# plt.plot(y_test, y_predict)
# plt.scatter(this_X, y, s=3)


from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
print classification_report(y_test,y_predict)
print confusion_matrix(y_test, y_predict)
#print (tn, fp, fn, tp)
```