MINOR PROJECT SEM3

AMRITA CHAUDHURI

AGENDA

Courses completed

The course instructors

Concepts learnt

Assessments

Demo project based on skills learnt through Coursera

INTRODUCTION

As a part of 25hr video course for doing the mini project in MCA AI Sem3 , below courses I selected and tried to complete.

Structuring Machine Learning Projects by DeepLearning.AI Andrew NG

OBJECTIVE :

- Explain why Machine Learning strategy is important
- Apply satisficing and optimizing metrics to set up your goal for ML projects
- Choose a correct train/dev/test split of your dataset
- Define human-level performance
- Use human-level performance to define key priorities in ML projects
- Take the correct ML Strategic decision based on observations of performances and dataset

Fundamentals of Scalable Data Science IBM

- Describe the challenges of data analytics
- Describe different methods used for IoT data analysis
- Create and deploy a test data generator capable of simulating IoT sensor data coming from a hypothetical washing machine
- Apply a solution that captures and stores IoT data from connected devices with Node-Red and Apache ChouchDB NoSQL
- Show how to process large data with ApacheSpark
- Describe different statistical measures (moments) used in summarizing data
- Describe how to process large amount of data arriving in high velocity by using ApacheSpark and SQL
- Explain the concept of multi-dimensional vector spaces and how any type of data corpus can be understood as points in that space
- Illustrate transformation and basic visualization of data

CONTINUED...

•Demonstrate how to plot Diagrams of low dimensional data sets like Box Plot, Run Chart, Scatter Plot and Histogram
•Analyze and draw conclusions out of the diagrams you've plotted
•Analyze and reduce dimensions of your data set

- **Advanced Machine Learning and Signal Processing IBM**

- Linear algebra , ML pipeline , statistics , Regression , clustering, SVM, K means

- **Applied AI with DeepLearning**

Demonstrate the ability to execute a saved Keras Model
•Recognize the projects within DeepLearning4J
•Choose the proper Framework for DeepLearning
Describe the tools used to get a model from Keras into DL4J

INSTRUCTORS

Andrew Ng is Founder of DeepLearning.AI, General Partner at AI Fund, Chairman and Co-Founder of Coursera, and an Adjunct Professor at Stanford University.

Romeo Kienzler Chief Data Scientist and course lead, holds a M. Sc. (ETH) in Information Systems, Bioinformatics & Applied Statistics (Swiss Federal Institute of Technology). He has nearly two decades of experience in Software Enineering, Database Administration and Information Integration. Since 2012 he works as a Data Scientist for IBM.

Nikolay Manchev holds an MSc in Software Technologies, an MSc in Data Science, and is currently undertaking postgraduate research at King's College London. His area of expertise is Machine Learning and Data Science, and his research interests are in neural networks with emphasis on biological plausibility.

Concepts learnt

OVERVIEW OF IOT DATA ANALYSIS

DIFFERENT ML ALGORITHMS BOTH SUPERVISED AND UNSUPERVISED

NEURAL NETWORK

LINEAR ALGEBRA AND STATISTICS

DATA PREPROCESSING

HYPERPARAMETER TUNING

ASSESSMENTS

- Generate data with IoT data storage calculator for IBM cloud-->create a test data generator using Node-red→
- Publishing data to Watson IOT platform-->Implement a flow to subscribe to this data and store it in Nosql DB[Apache CouchDB(Cloudant)] --> creating a function which is used to create a data frame from a cloudant data frame using the "DataSource" which is some sort of a plugin which allows ApacheSpark to use different data sources.-->then passed the dataframe object. also registered the dataframe in the ApacheSparkSQL catalog --> issue queries against the "washing" table using "spark.sql()-->implement a function which returns a (python) list of string values of the field names in this data frame--> connect to the cloudant database --> test the created functions

- For each module separate lab was there both for practice and graded submission
- **1**. generate sensor data-->https://github.com/IBM/coursera/raw/master/coursera_ml/a2.parquet---> analysing the data using spark-->creating a VectorAssembler which consumes columns X, Y and Z and produces a column “features”\n"--> instantiate a classifier from the SparkML package and assign it to the classifier variable-->Rename the “CLASS” column to “label”-->Specify the label-column correctly to be “CLASS”-->train and evaluate-->submit.

- 2. Again load parquet file-->check if we have balanced classes – this means that we have roughly the same number of examples for each class we want to predict which is also important for classification and clustering -->create a VectorAssembler which consumes columns X, Y and Z and produces a column “features”--> insatiate a clustering algorithm from the SparkML package and assign it to the clust variable-->from clustering import Kmeans-->from pyspark import pipeline--> fit the model and predict

DEMO PROJECT BASED ON SKILLS LEARNT

Mammography is the most effective method for breast cancer screening available today. However, the low positive predictive value of breast biopsy resulting from mammogram interpretation leads to approximately 70% unnecessary biopsies with benign outcomes.

This data set can be used to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age. It contains a BI-RADS assessment, the patient's age and three BI-RADS attributes together with the ground truth (the severity field) for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the Institute of Radiology of the University Erlangen-Nuremberg between 2003 and 2006.

https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass

This data contains 961 instances of masses detected in mammograms, and contains the following attributes:

1.BI-RADS assessment: 1 to 5 (ordinal)

2.Age: patient's age in years (integer)

3.Shape: mass shape: round=1 oval=2 lobular=3 irregular=4 (nominal)

4.Margin: mass margin: circumscribed=1 microlobulated=2 obscured=3 ill-defined=4 spiculated=5 (nominal)

5.Density: mass density high=1 iso=2 low=3 fat-containing=4 (ordinal)

6.Severity: benign=0 or malignant=1 (binominal)

BI-RADS is an assesment of how confident the severity classification is; it is not a "predictive" attribute and so we will discard it. The age, shape, margin, and density attributes are the features that we will build our model with, and "severity" is the classification we will attempt to predict based on those attributes.

Although "shape" and "margin" are nominal data types, which sklearn typically doesn't deal with well, they are close enough to ordinal that we shouldn't just discard them. The "shape" for example is ordered increasingly from round to irregular.

A lot of unnecessary anguish and surgery arises from false positives arising from mammogram results. If we can build a better way to interpret them through supervised machine learning, it could improve a lot of lives.

Apply several different supervised machine learning techniques to this data set, and see which one yields the highest accuracy as measured with K-Fold cross validation (K=10). Apply:

•Decision tree

•Random forest

•KNN

•Naive Bayes

•SVM

•Logistic Regression

•a neural network using Keras.

The data needs to be cleansed; many rows contain missing data, and there may be erroneous data identifiable as outliers as well.

For SVM it is required the input data to be normalized first.

Start by importing the mammographic_masses.data.txt file into a Pandas dataframe (hint: use read_csv) and take a look at it.

```
]: import pandas as pd

masses_data = pd.read_csv('mammographic_masses.data.txt')
masses_data.head()
```

]:

	5	67	3	5.1	3.1	1
0	4	43	1	1	?	1
1	5	58	4	5	3	1
2	4	28	1	1	3	0
3	5	74	1	5	?	1
4	4	65	1	?	3	0

using the optional parmaters in read_csv to convert missing data (indicated by a ?) into NaN, and to add the appropriate column names (BI_RADS, age, shape, margin, density, and severity):

In [2]:
```
masses_data = pd.read_csv('mammographic_masses.data.txt', na_values=['?'], names = ['BI-RADS', 'age', 'shape', 'margin', 'density
masses_data.head()
```

Out[2]:

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
1	4.0	43.0	1.0	1.0	NaN	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
4	5.0	74.0	1.0	5.0	NaN	1

Evaluate whether the data needs cleaning; your model is only as good as the data it's given. Hint: use describe() on the dataframe.

In [3]:
```
masses_data.describe()
```

Out[3]:

	BI-RADS	age	shape	margin	density	severity
count	959.000000	956.000000	930.000000	913.000000	885.000000	961.000000
mean	4.348279	55.487448	2.721505	2.796276	2.910734	0.463059
std	1.783031	14.480131	1.242792	1.566546	0.380444	0.498893
min	0.000000	18.000000	1.000000	1.000000	1.000000	0.000000
25%	4.000000	45.000000	2.000000	1.000000	3.000000	0.000000
50%	4.000000	57.000000	3.000000	3.000000	3.000000	0.000000

There are quite a few missing values in the data set. Before we just drop every row that's missing data, need to make sure not to bias our data in doing so. need to check if any sort of correlation is present. and what sort of data has missing fields?

In [4]:
```
masses_data.loc[(masses_data['age'].isnull()) |
                (masses_data['shape'].isnull()) |
                (masses_data['margin'].isnull()) |
                (masses_data['density'].isnull())]
```

Out[4]:

	BI-RADS	age	shape	margin	density	severity
1	4.0	43.0	1.0	1.0	NaN	1
4	5.0	74.0	1.0	5.0	NaN	1
5	4.0	65.0	1.0	NaN	3.0	0
6	4.0	70.0	NaN	NaN	3.0	0
7	5.0	42.0	1.0	NaN	3.0	0
...	...	...	...	...	...	...
778	4.0	60.0	NaN	4.0	3.0	0
819	4.0	35.0	3.0	NaN	2.0	0
824	6.0	40.0	NaN	3.0	4.0	1
884	5.0	NaN	4.0	4.0	3.0	1
923	5.0	NaN	4.0	3.0	3.0	1

130 rows × 6 columns

If the missing data seems randomly distributed, then drop rows with missing data , using dropna().

In [5]:
```
masses_data.dropna(inplace=True)
masses_data.describe()
```

Out[5]:

	BI-RADS	age	shape	margin	density	severity
count	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000
mean	4.393976	55.781928	2.781928	2.813253	2.915663	0.485542
std	1.888371	14.671782	1.242361	1.567175	0.350936	0.500092
min	0.000000	18.000000	1.000000	1.000000	1.000000	0.000000
25%	4.000000	46.000000	2.000000	1.000000	3.000000	0.000000
50%	4.000000	57.000000	3.000000	3.000000	3.000000	0.000000
75%	5.000000	66.000000	4.000000	4.000000	3.000000	1.000000
max	55.000000	96.000000	4.000000	5.000000	4.000000	1.000000

Next you'll need to convert the Pandas dataframes into numpy arrays that can be used by scikit_learn. Create an array that extracts only the feature data we want to work with (age, shape, margin, and density) and another array that contains the classes (severity). You'll also need an array of the feature name labels.

In [6]:
```
all_features = masses_data[['age', 'shape',
                            'margin', 'density']].values


all_classes = masses_data['severity'].values

feature_names = ['age', 'shape', 'margin', 'density']

all_features
```

Out[6]:
```
array([[67.,  3.,  5.,  3.],
       [58.,  4.,  5.,  3.],
```

```
feature_names = ['age', 'shape', 'margin', 'density']

all_features
```

Out[6]:
```
array([[67.,  3.,  5.,  3.],
       [58.,  4.,  5.,  3.],
       [28.,  1.,  1.,  3.],
       ...,
       [64.,  4.,  5.,  3.],
       [66.,  4.,  5.,  3.],
       [62.,  3.,  3.,  3.]])
```

Some of the models require the input data to be normalized, hence using preprocessing.StandardScaler().

In [7]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(all_features)
all_features_scaled
```

Out[7]:
```
array([[ 0.7650629 ,  0.17563638,  1.39618483,  0.24046607],
       [ 0.15127063,  0.98104077,  1.39618483,  0.24046607],
       [-1.89470363, -1.43517241, -1.157718  ,  0.24046607],
       ...,
       [ 0.56046548,  0.98104077,  1.39618483,  0.24046607],
       [ 0.69686376,  0.98104077,  1.39618483,  0.24046607],
       [ 0.42406719,  0.17563638,  0.11923341,  0.24046607]])
```

```
scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(all_features)
all_features_scaled
```

Out[7]:
```
array([[ 0.7650629 ,  0.17563638,  1.39618483,  0.24046607],
       [ 0.15127063,  0.98104077,  1.39618483,  0.24046607],
       [-1.89470363, -1.43517241, -1.157718  ,  0.24046607],
       ...,
       [ 0.56046548,  0.98104077,  1.39618483,  0.24046607],
       [ 0.69686376,  0.98104077,  1.39618483,  0.24046607],
       [ 0.42406719,  0.17563638,  0.11923341,  0.24046607]])
```

Decision Trees

Before moving to K-Fold cross validation and random forests, start by creating a single train/test split of our data. Set aside 75% for training, and 25% for testing.

In [8]:
```
import numpy
from sklearn.model_selection import train_test_split

numpy.random.seed(1234)

(training_inputs,
 testing_inputs,
 training_classes,
 testing_classes) = train_test_split(all_features_scaled, all_classes, train_size=0.75, random_state=1)
```

Now create a DecisionTreeClassifier and fit it to your training data.

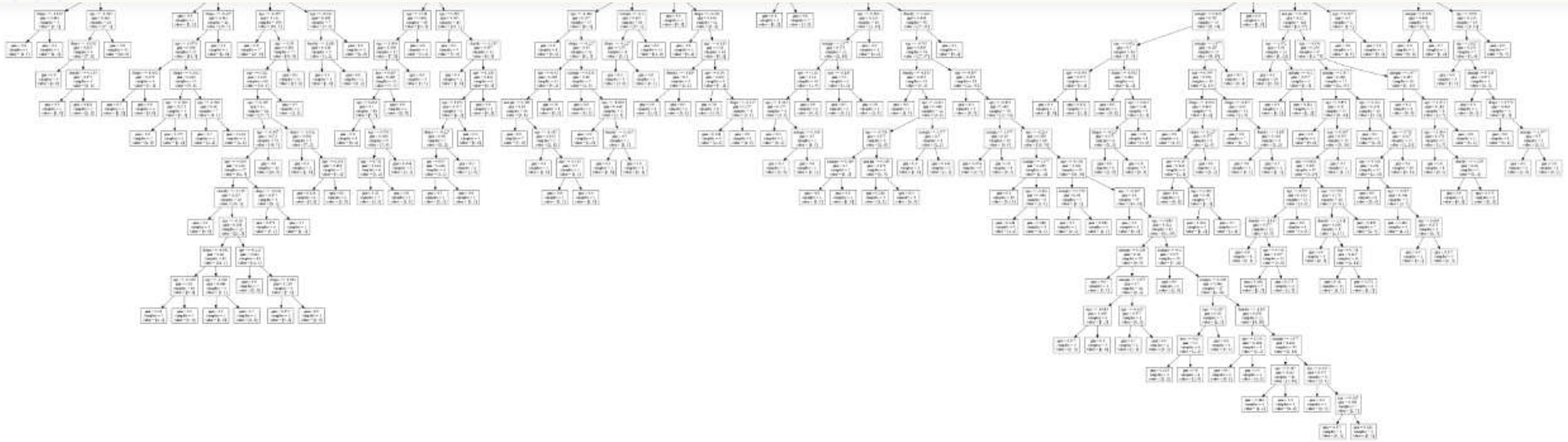
In [9]:
```
from sklearn.tree import DecisionTreeClassifier

clf= DecisionTreeClassifier(random_state=1)
```

In [9]:
```
from sklearn.tree import DecisionTreeClassifier

clf= DecisionTreeClassifier(random_state=1)

# Train the classifier on the training set
clf.fit(training_inputs, training_classes)
```

Out[9]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1, splitter='best')
```

Display the resulting decision tree.

In [10]:
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn import tree
from pydotplus import graph_from_dot_data

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                         feature_names=feature_names)
graph = graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



Measuring the accuracy of the resulting decision tree model using the test data.

In [11]:
```
clf.score(testing_inputs, testing_classes)
```

Out[11]:
```
0.7355769230769231
```

Now instead of a single train/test split, using K-Fold cross validation to get a better measure of model's accuracy (K=10).
using model_selection.cross_val_score

In [12]:
```
from sklearn.model_selection import cross_val_score

clf = DecisionTreeClassifier(random_state=1)

cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)
```

In [12]:
```
from sklearn.model_selection import cross_val_score

clf = DecisionTreeClassifier(random_state=1)

cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)

cv_scores.mean()
```

Out[12]:
```
0.7373123154639465
```

Now try a RandomForestClassifier instead. Does it perform better?

In [13]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=10, random_state=1)
cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)

cv_scores.mean()
```

Out[13]:
```
0.7528157927878762
```

SVM

using svm.SVC with a linear kernel.

In [14]:
```
from sklearn import svm

C = 1.0
svc = svm.SVC(kernel='linear', C=C)
```

In [15]:
```
cv_scores = cross_val_score(svc, all_features_scaled, all_classes, cv=10)

cv_scores.mean()
```

Out[15]: 0.7964988875362076

KNN-using neighbors.KNeighborsClassifier - Starting with a K of 10. K is an example of a hyperparameter - a parameter on the model itself which may need to be tuned for best results on a particular data set.

In [16]:
```
from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(n_neighbors=10)
cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)
```

In [16]:
```
from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(n_neighbors=10)
cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)

cv_scores.mean()
```

Out[16]:
```
0.7854795488574507
```

As choosing K is a bit tricky, so trying different values of K. Writing a for loop to run KNN with K values ranging from 1 to 50 and see if K makes a substantial difference.

In [17]:
```
for n in range(1, 50):
    clf = neighbors.KNeighborsClassifier(n_neighbors=n)
    cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)
    print (n, cv_scores.mean())
```

```
1 0.7239123742356184
2 0.6889838098036746
3 0.7541080699103032
4 0.7300813008130081
5 0.7735464506108056
6 0.7626163189342738
7 0.7940595133145824
8 0.7747082406280172
9 0.7880200243482641
10 0.7854795488574507
11 0.7915333809104012
12 0.7794257168045002
13 0.7819084701174035
```

Naive Bayes

Now trying naive_bayes.MultinomialNB.

[18]:
```
from sklearn.naive_bayes import MultinomialNB

scaler = preprocessing.MinMaxScaler()
all_features_minmax = scaler.fit_transform(all_features)

clf = MultinomialNB()
cv_scores = cross_val_score(clf, all_features_minmax, all_classes, cv=10)

cv_scores.mean()
```

t[18]: 0.7844055665169388

Revisiting SVM

svm.SVC may perform differently with different kernels. The choice of kernel is an example of a "hyperparamter." Trying the rbf, sigmoid, and poly kernels and see what the best-performing kernel is.

[19]:
```
C = 1.0
svc = svm.SVC(kernel='rbf', C=C)
cv_scores = cross_val_score(svc, all_features_scaled, all_classes, cv=10)
cv_scores.mean()
```

Out[21]: 0.792753942599667

Logistic Regression

fundamentally this is just a binary classification problem.

In [22]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
cv_scores = cross_val_score(clf, all_features_scaled, all_classes, cv=10)
cv_scores.mean()
```

```
  FutureWarning)
E:\Anaconda3\lib\site-packages\sklear
n 0.22. Specify a solver to silence t
  FutureWarning)
```

[22]: 0.8073583532737221

Neural Networks

trying artificial neural network atlast.. using Keras to set up a neural network with 1 binary output neuron and see how it performs.

23]:
```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

def create_model():
    model = Sequential()
    #4 feature inputs going into an 6-unit layer (more does not seem to help - in fact you can go down to 4)
    model.add(Dense(6, input_dim=4, kernel_initializer='normal', activation='relu'))
    # "Deep learning" turns out to be unnecessary - this additional hidden layer doesn't help either.
    #model.add(Dense(4, kernel_initializer='normal', activation='relu'))
    # Output layer with a binary classification (benign or malignant)
    model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
    # Compile model; adam seemed to work best
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

24]:
```
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

# Wrap our Keras model in an estimator compatible with scikit_learn
estimator = KerasClassifier(build_fn=create_model, epochs=100, verbose=0)
# Now we can use scikit_learn's cross_val_score to evaluate this model identically to the others
cv_scores = cross_val_score(estimator, all_features_scaled, all_classes, cv=10)
cv_scores.mean()
```

24]: 0.8024096429347992

```
    return model
```

In [24]:
```
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

# Wrap our Keras model in an estimator compatible with scikit_learn
estimator = KerasClassifier(build_fn=create_model, epochs=100, verbose=0)
# Now we can use scikit_learn's cross_val_score to evaluate this model identically to the others
cv_scores = cross_val_score(estimator, all_features_scaled, all_classes, cv=10)
cv_scores.mean()
```

Out[24]:
```
0.8024096429347992
```

```
BEST MODEL in terms of accuracy....
```

except decision trees , other algorithms could be tuned to produce comparable results with 79-80% accuracy.

```
Additional hyperparameter tuning, or different topologies of the multi-level perceptron might make a difference.will continue to work on it.
```