ranaatul61 > Machine Learning >

# Deep Learning

## MLP Classification with a Kaggle Problem

posted Feb 1, 2018, 10:57 PM by Atul Rana **[ updated Feb 1, 2018, 11:31 PM ]**

Multi-Layer Perceptron Classifier

## Let's try to solve a Kaggle Problem "Poker Rule Induction".

```
#import libraries
import pandas as pd
```

### Getting the data

```
sampleSub = pd.read_csv("sampleSubmission.csv")
test = pd.read_csv('test.csv')
train = pd.read_csv('train.csv')
```

Making our feature and prediction data seperate.

```
x = train.drop('hand',axis=1)
y = train['hand']
```

```
test.head()
```

| | id | S1 | C1 | S2 | C2 | S3 | C3 | S4 | C4 | S5 | C5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 10 | 2 | 2 | 3 | 3 | 3 | 8 | 1 | 1 |
| **1** | 2 | 2 | 13 | 3 | 5 | 3 | 7 | 4 | 6 | 1 | 4 |
| **2** | 3 | 1 | 3 | 1 | 11 | 2 | 8 | 2 | 1 | 2 | 4 |
| **3** | 4 | 1 | 6 | 3 | 3 | 4 | 7 | 1 | 8 | 3 | 11 |
| **4** | 5 | 2 | 10 | 3 | 4 | 1 | 6 | 2 | 12 | 2 | 6 |

```
x.head()
```

| | S1 | C1 | S2 | C2 | S3 | C3 | S4 | C4 | S5 | C5 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 10 | 2 | 2 | 3 | 3 | 3 | 8 | 1 | 1 |
| **1** | 2 | 13 | 3 | 5 | 3 | 7 | 4 | 6 | 1 | 4 |
| **2** | 1 | 3 | 1 | 11 | 2 | 8 | 2 | 1 | 2 | 4 |
| **3** | 1 | 6 | 3 | 3 | 4 | 7 | 1 | 8 | 3 | 11 |
| **4** | 2 | 10 | 3 | 4 | 1 | 6 | 2 | 12 | 2 | 6 |

```
y.head()
```

```
0    0
1    0
2    2
3    3
4    0
```

### Train-Test Splitting

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

Definning the Model with (200,150,100) hidden layer node sizes and activation function as "relu"

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes=(200,150,100),activation='relu')
```

### Fitting the training data to the model

```
classifier.fit(X_train,y_train)
```

default parameters looks like:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(200, 150, 100), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

### Prediction

```
pred = classifier.predict(X_test)
```

### Model Evaluation

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

```
classifier.score(X_test, y_test)
```

```
    0.92502998800479808
92% Accuracy is not that bad for starting.
```

```
print(classification_report(pred,y_test))
```

```
             precision    recall  f1-score   support

          0       0.99      0.94      0.96      2615
          1       0.93      0.93      0.93      2167
          2       0.46      0.75      0.57       141
          3       0.59      0.78      0.67        73
          4       0.08      0.40      0.13         5
          5       0.00      0.00      0.00         0
          6       0.00      0.00      0.00         0
          7       1.00      1.00      1.00         1
          8       0.00      0.00      0.00         0
          9       0.00      0.00      0.00         0

avg / total       0.94      0.93      0.93      5002
```

```
print(confusion_matrix(pred,y_test))
```

```
    [[2455  134    0    0   15   10    0    0    0    1]
     [  27 2006  111   12    8    0    0    0    1    2]
     [   0    5  106   28    0    0    2    0    0    0]
     [   0    0   13   57    0    0    3    0    0    0]
     [   1    2    0    0    2    0    0    0    0    0]
     [   0    0    0    0    0    0    0    0    0    0]
     [   0    0    0    0    0    0    0    0    0    0]
     [   0    0    0    0    0    0    0    1    0    0]
     [   0    0    0    0    0    0    0    0    0    0]
     [   0    0    0    0    0    0    0    0    0    0]]
```

## Making Submission File

making test data suitable for prediction

```
Test = test.drop('id',axis=1)
Test.head()
```

|   | S1 | C1 | S2 | C2 | S3 | C3 | S4 | C4 | S5 | C5 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 10 | 2 | 2 | 3 | 3 | 3 | 8 | 1 | 1 |
| **1** | 2 | 13 | 3 | 5 | 3 | 7 | 4 | 6 | 1 | 4 |
| **2** | 1 | 3 | 1 | 11 | 2 | 8 | 2 | 1 | 2 | 4 |
| **3** | 1 | 6 | 3 | 3 | 4 | 7 | 1 | 8 | 3 | 11 |
| **4** | 2 | 10 | 3 | 4 | 1 | 6 | 2 | 12 | 2 | 6 |

```
training model with whole training data provided in the Kaggle Problem.
```

```
classifier.fit(x,y)
pred = classifier.predict(Test)
```

```
Making the submission file in the given format
```

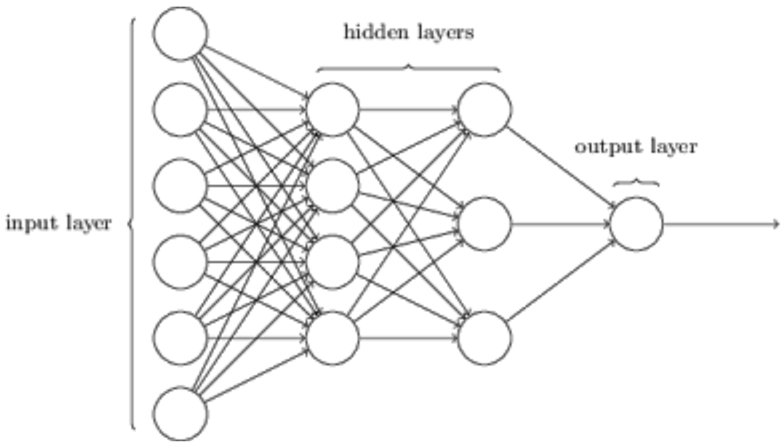
```
pred = pd.DataFrame(pred)
sampleSub['hand'] = pred
sampleSub.to_csv('sumbit.csv',index=False)
```

## Neural Networks

posted Jan 31, 2018, 1:10 AM by Atul Rana **[ updated 37 minutes ago ]**

Introduction

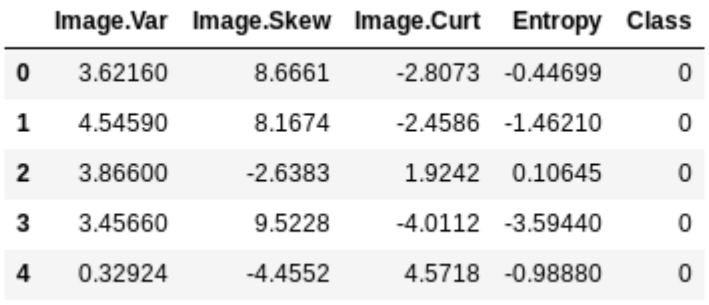
*Neural Network* is a powerful tool used in modern intelligent systems. Nowadays, many applications that involve pattern recognition, feature mapping, clustering, *classification* and etc. use Neural Networks as an essential component.



```
#using pandas to read .csv file
import pandas as pd
```

**import the data from CSV file**

```
data = pd.read_csv('bank_note_data.csv')
data.head()
```



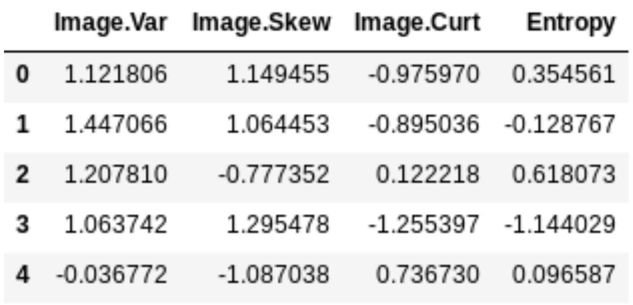
| | Image.Var | Image.Skew | Image.Curt | Entropy | Class |
|---|---|---|---|---|---|
| **0** | 3.62160 | 8.6661 | -2.8073 | -0.44699 | 0 |
| **1** | 4.54590 | 8.1674 | -2.4586 | -1.46210 | 0 |
| **2** | 3.86600 | -2.6383 | 1.9242 | 0.10645 | 0 |
| **3** | 3.45660 | 9.5228 | -4.0112 | -3.59440 | 0 |
| **4** | 0.32924 | -4.4552 | 4.5718 | -0.98880 | 0 |

When using Neural Network and Deep Learning based systems, it is usually a good idea to Standardize your data, this step isn't actually necessary for our particular data set, but let's run through it for practice!

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data.drop('Class',axis=1))
scaled_features = scaler.fit_transform(data.drop('Class',axis=1))
```

**See how our data looks like after feature scaling.**

```
X= pd.DataFrame(scaled_features,columns=data.columns[:-1])
X.head()
```



| | Image.Var | Image.Skew | Image.Curt | Entropy |
|---|---|---|---|---|
| **0** | 1.121806 | 1.149455 | -0.975970 | 0.354561 |
| **1** | 1.447066 | 1.064453 | -0.895036 | -0.128767 |
| **2** | 1.207810 | -0.777352 | 0.122218 | 0.618073 |
| **3** | 1.063742 | 1.295478 | -1.255397 | -1.144029 |
| **4** | -0.036772 | -1.087038 | 0.736730 | 0.096587 |

```
y = data['class']
```

```
X.shape
```

```
(1372, 4)
```

```
X = X.as_matrix()
```

```
y = y.as_matrix()
```

**Use the .as_matrix() method on X and Y and reset them equal to this result. We need to do this in order for TensorFlow to accept the data in Numpy array form instead of a pandas series.**

**Train Test splitting the Data.**

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

### Contrib.learn

**importing tensorflow.contrib.learn as learn**

```
import tensorflow.contrib.learn.python.learn as learn
```

```
feature_columns = learn.infer_real_valued_columns_from_input(X)
feature_columns
```

```
[_RealValuedColumn(column_name='', dimension=4, default_value=None, dtype=tf.float64,
    normalizer=None)]
```

**Creating an object called classifier which is a DNNClassifier from learn. Setting it to have 2 classes and a [10,20,10] hidden unit layer structure.**

```
classifier = learn.DNNClassifier(hidden_units=[10, 20, 10], n_classes=2)
```

### Fitting data to classifier and make prediction for X_test

Fitting the data to the classifier. Use steps 200 with batch_size of 20. You can play around with these values depending upon your machine limits.

*Note: Ignore any warnings you get, they won't affect your output*

```
classifier.fit(X_train, y_train, steps=200, batch_size=20)
```

```
note_predictions = classifier.predict(X_test)
```

## Model Evaluation

**import metrics**

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

```
print(classifier.evaluate(X_test,y_test)["accuracy"])
```

1.0

```
from numpy import array
pre = array( list(note_predictions))
```

```
print(confusion_matrix(y_test,pre))
```

```
    [[218   0]
     [  0 194]]
```

```
print(classification_report(y_test,pre))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       218
          1       1.00      1.00      1.00       194

avg / total       1.00      1.00      1.00       412
```

```
Yeah!! 100% Accuracy..
```

1-2 of 2 ‹ ›