Deep Learning using Tensorflow

January 8, 2021

0.1 Deep Learning using Tensorflow

This analysis is done using the Bank Authentication Data Set from the UCI repository.

The data consists of 5 columns:

- variance of Wavelet Transformed image (continuous)
- skewness of Wavelet Transformed image (continuous)
- curtosis of Wavelet Transformed image (continuous)
- entropy of image (continuous)
- class (integer)

Where class indicates whether or not a Bank Note was authentic.

[1]:
```python
import pandas as pd
```

[2]:
```python
data = pd.read_csv('bank_note_data.csv')
```

[3]:
```python
data.head()
```

[3]:

	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

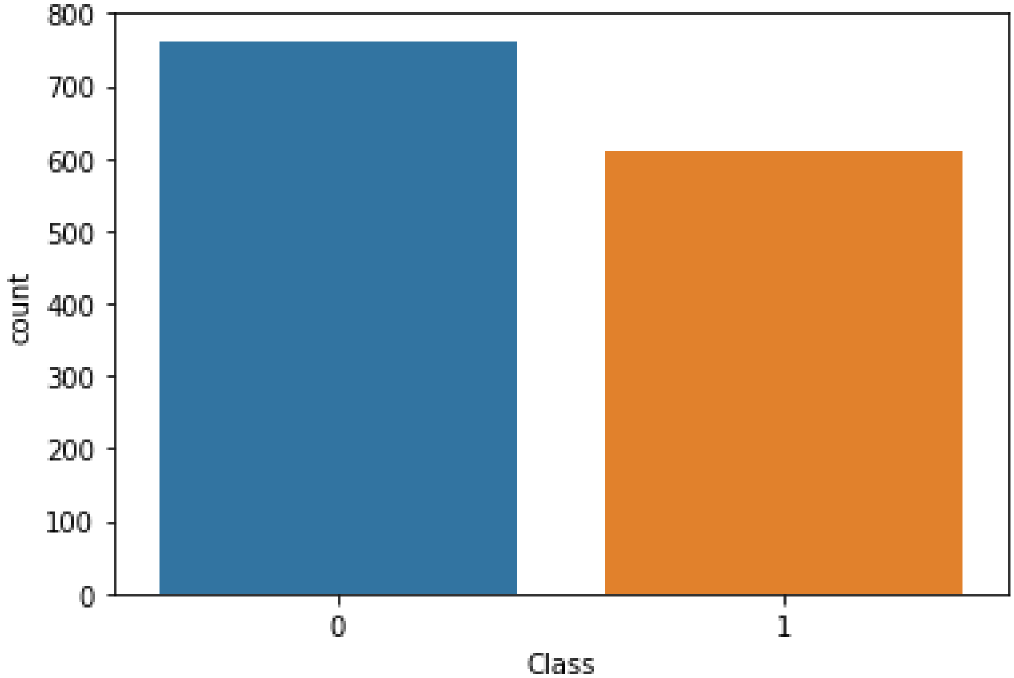
0.1.1 Exploratory Data Analysis

[4]:
```python
import seaborn as sns
%matplotlib inline
```

Countplot of the Classes (Authentic 1 vs Fake 0)

[5]:
```python
sns.countplot(x='Class',data=data)
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26bb34edda0>
```

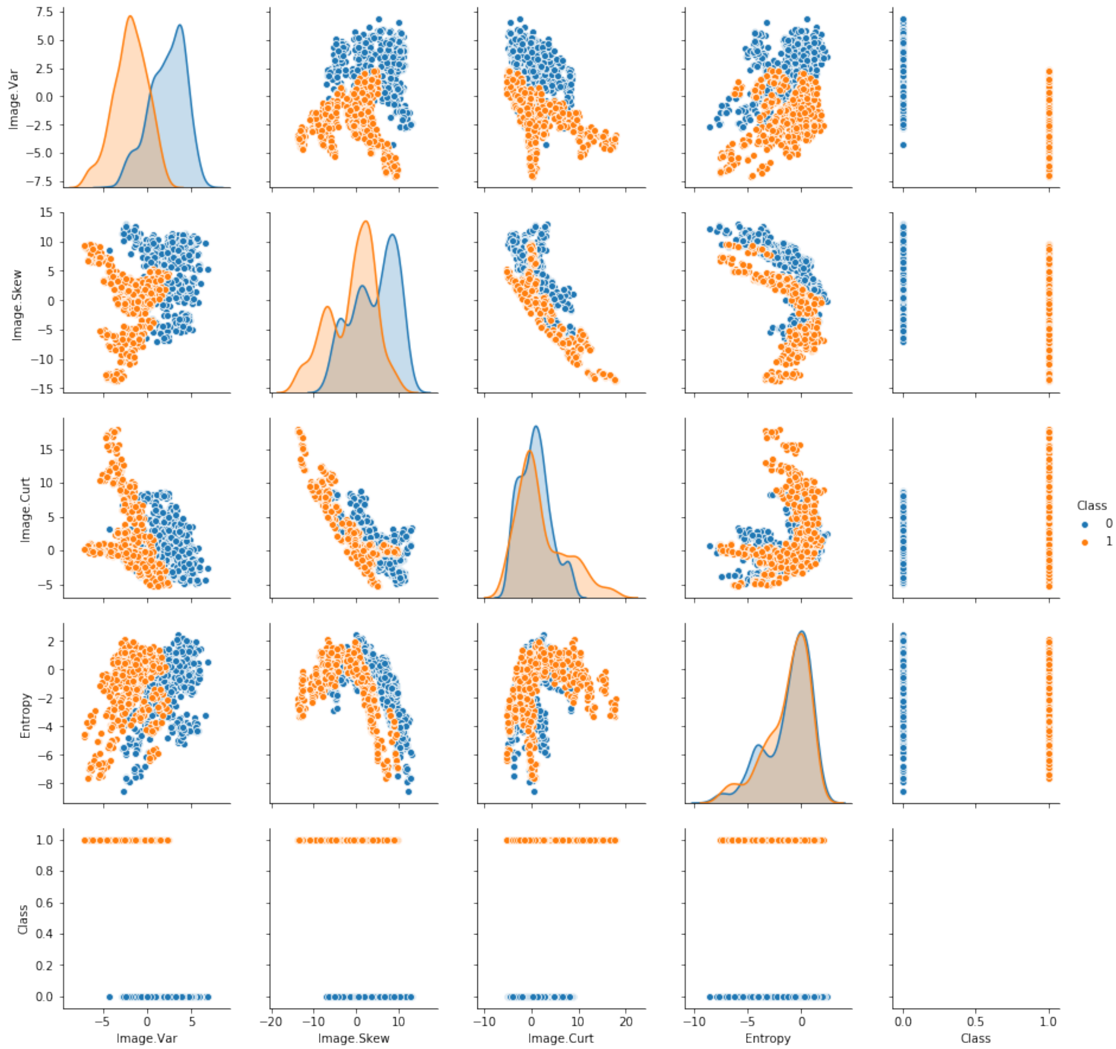


PairPlot of the Data using Seaborn with Hue = Class

[6]:
```
sns.pairplot(data,hue='Class')
```

```
C:\Users\Marcial\Anaconda3\lib\site-
packages\statsmodels\nonparametric\kde.py:494: RuntimeWarning: invalid value
encountered in true_divide
  binned = fast_linbin(X,a,b,gridsize)/(delta*nobs)
C:\Users\Marcial\Anaconda3\lib\site-
packages\statsmodels\nonparametric\kdetools.py:34: RuntimeWarning: invalid value
encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
C:\Users\Marcial\Anaconda3\lib\site-packages\numpy\core\_methods.py:26:
RuntimeWarning: invalid value encountered in reduce
  return umr_maximum(a, axis, None, out, keepdims)
```

[6]:
```
<seaborn.axisgrid.PairGrid at 0x26bb34c9da0>
```



0.1.2 Data Preparation

Standard Scaling

[7]:
```
from sklearn.preprocessing import StandardScaler
```

[8]:
```
scaler = StandardScaler()
```

Fit scaler to the features.

[9]:
```
scaler.fit(data.drop('Class',axis=1))
```

[9]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Use the .transform() method to transform the features to a scaled version.

[10]:
```
scaled_features = scaler.fit_transform(data.drop('Class',axis=1))
```

Convert the scaled features to a dataframe and check the head of this dataframe to make sure the scaling worked.

[11]:
```
df_feat = pd.DataFrame(scaled_features,columns=data.columns[:-1])
df_feat.head()
```

[11]:

	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

0.1.3 Train Test Split

[12]:
```
X = df_feat
```

[13]:
```
y = data['Class']
```

[14]:
```
from sklearn.model_selection import train_test_split
```

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

0.1.4 Tensorflow

[16]:
```
import tensorflow as tf
```

```
C:\Users\Marcial\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning:
Conversion of the second argument of issubdtype from `float` to `np.floating` is
deprecated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Create a list of feature column objects using tf.feature.numeric_column()

[17]:
```
df_feat.columns
```

[17]:
```
Index(['Image.Var', 'Image.Skew', 'Image.Curt', 'Entropy'], dtype='object')
```

[18]:
```
image_var = tf.feature_column.numeric_column("Image.Var")
image_skew = tf.feature_column.numeric_column('Image.Skew')
image_curt = tf.feature_column.numeric_column('Image.Curt')
entropy =tf.feature_column.numeric_column('Entropy')
```

[19]:
```
feat_cols = [image_var,image_skew,image_curt,entropy]
```

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

[20]:
```
classifier = tf.estimator.DNNClassifier(hidden_units=[10, 20, 10],␣
↪n_classes=2,feature_columns=feat_cols)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory:
C:\Users\Marcial\AppData\Local\Temp\tmpw8v7z_z6
INFO:tensorflow:Using config: {'_model_dir':
'C:\\Users\\Marcial\\AppData\\Local\\Temp\\tmpw8v7z_z6', '_tf_random_seed':
None, '_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': None, '_keep_checkpoint_max':
5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_device_fn': None, '_service': None,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at
0x0000026BBA0F9FD0>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

Now create a tf.estimator.pandas_input_fn that takes in your X_train, y_train with batch_size = 20 and set shuffle=True.

[21]:
```
input_func = tf.estimator.inputs.
↪pandas_input_fn(x=X_train,y=y_train,batch_size=20,shuffle=True)
```

Train the classifier to the input function using steps=500.

[22]:
```
classifier.train(input_fn=input_func,steps=500)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into
C:\Users\Marcial\AppData\Local\Temp\tmpw8v7z_z6\model.ckpt.
INFO:tensorflow:loss = 13.792015, step = 1
INFO:tensorflow:Saving checkpoints for 48 into
C:\Users\Marcial\AppData\Local\Temp\tmpw8v7z_z6\model.ckpt.
INFO:tensorflow:Loss for final step: 0.47980386.
```

[22]:
```
<tensorflow.python.estimator.canned.dnn.DNNClassifier at 0x26bbacc7c18>
```

0.2 Model Evaluation

Create another pandas_input_fn that takes in the X_test data for x. Set shuffle=False since there is no need to shuffle for predictions.

[23]:
```
pred_fn = tf.estimator.inputs.
 ↪pandas_input_fn(x=X_test,batch_size=len(X_test),shuffle=False)
```

Use the predict method from the classifier model to create predictions from X_test

[24]:
```
note_predictions = list(classifier.predict(input_fn=pred_fn))
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
C:\Users\Marcial\AppData\Local\Temp\tmpw8v7z_z6\model.ckpt-48
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

[25]:
```
note_predictions[0]
```

[25]:
```
{'class_ids': array([0], dtype=int64),
 'classes': array([b'0'], dtype=object),
 'logistic': array([0.00157453], dtype=float32),
 'logits': array([-6.4522204], dtype=float32),
 'probabilities': array([0.9984255 , 0.00157453], dtype=float32)}
```

[26]:
```
final_preds  = []
for pred in note_predictions:
    final_preds.append(pred['class_ids'][0])
```

Creating a classification report and a Confusion Matrix.

[27]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[28]:
```
print(confusion_matrix(y_test,final_preds))
```

```
[[213   2]
 [ 10 187]]
```

[29]:
```
print(classification_report(y_test,final_preds))
```

```
             precision    recall  f1-score   support

          0       0.96      0.99      0.97       215
          1       0.99      0.95      0.97       197

avg / total       0.97      0.97      0.97       412
```