BankNote_Authentication

September 12, 2020

1 Kaushal Rao - Bank Note Authentication (with ANNs)

For this project, we'll use the Bank Authentication Data Set from the UCI repository, which contains image data from bank notes and whether or not the bank note was authentic (denoted by 0 or 1 in the "class" column).

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)

We will be using a neural network (deep learning) to predict (binary classification) if a particular bank note is authentic!

1.1 The Data

```
In [1]: import pandas as pd
        data = pd.read_csv('bank_note_data.csv')

In [2]: data.head()
        # features and label column ('class')
```

Out[2]:

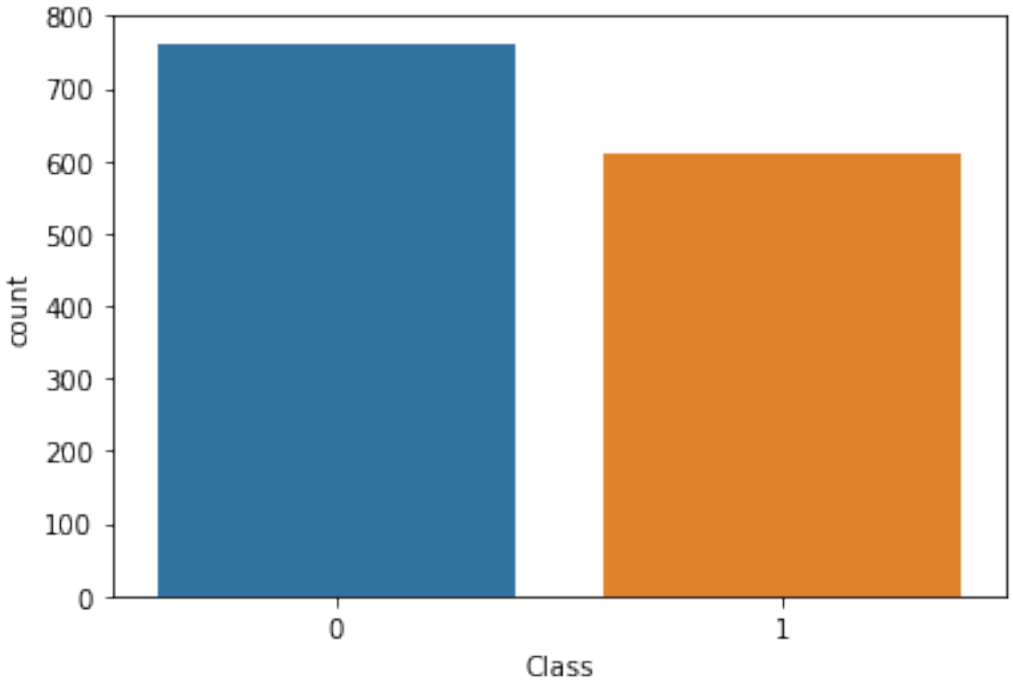
	Image.Var	Image.Skew	Image.Curt	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

1.2 Exploratory Data Analysis (EDA)

```
In [3]: import seaborn as sns
        %matplotlib inline

        sns.countplot(x='Class',data=data)
        # countplot of the classes (fake 0 vs authentic 1)
```

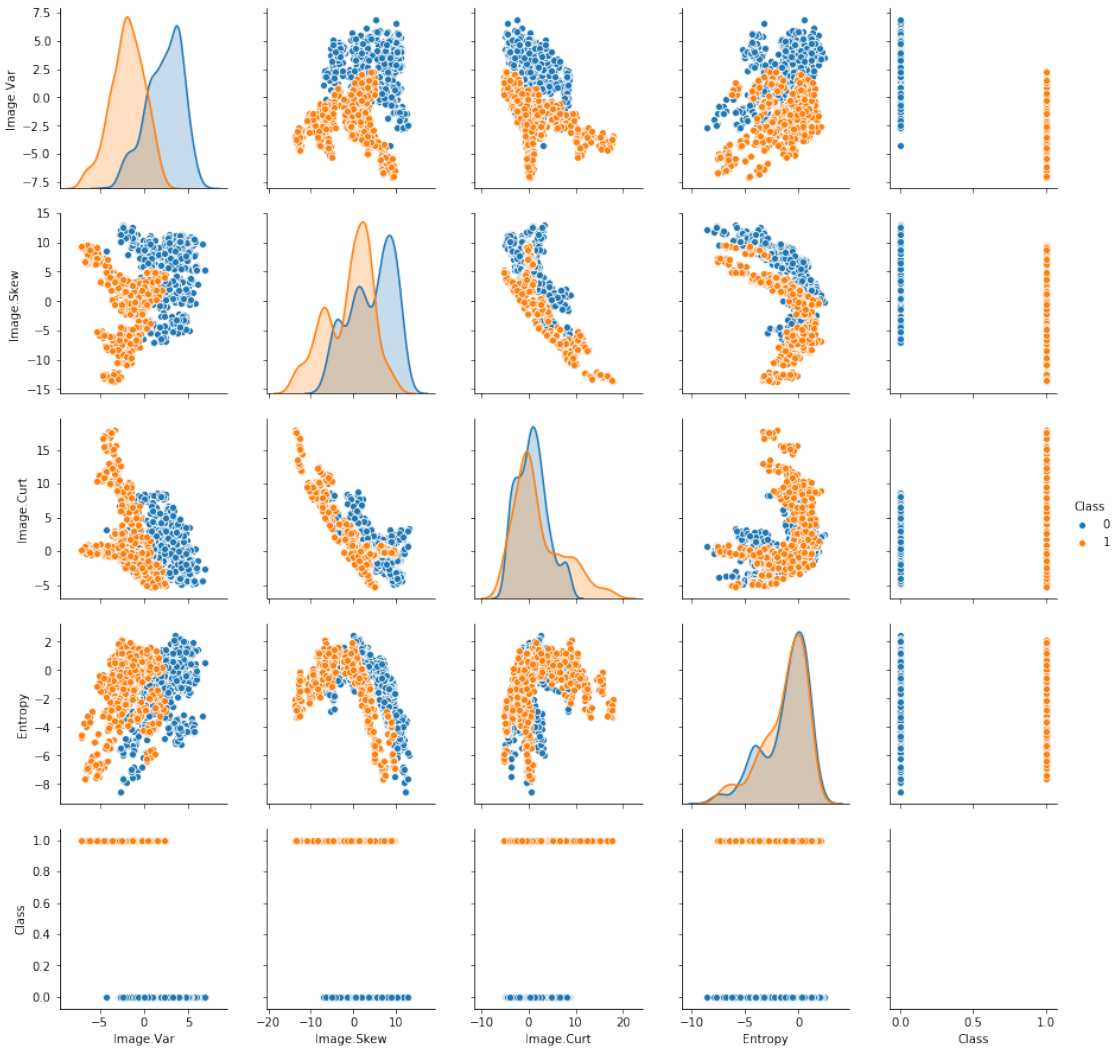
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f10e4e0>



In [4]:
```
sns.pairplot(data,hue='Class')
        # pairplot of the data, setting hue to the 'class' column
        # can see some differences in feature values between the classes
```

```
/Users/kaushalrao/anaconda3/lib/python3.7/site-packages/statsmodels/nonparametric/kde.py:488: I
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/Users/kaushalrao/anaconda3/lib/python3.7/site-packages/statsmodels/nonparametric/kdetools.py:3
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

Out[4]: <seaborn.axisgrid.PairGrid at 0x1a1f3c2f28>



1.3 Data Preprocessing & Standard Scaling

```
In [5]: from sklearn.preprocessing import StandardScaler

        scaler = StandardScaler()
        # creating StandardScaler() object
        scaler.fit(data.drop('Class',axis=1))
        # fitting scaler to the features of the data
        scaled_features = scaler.fit_transform(data.drop('Class',axis=1))
        # transforming features to the scaled version

In [6]: df_feat = pd.DataFrame(scaled_features,columns=data.columns[:-1])
        # converting array to dataframe
        df_feat.head()
        # feature values are scaled now
```

Out[6]:

	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

1.4 Train-Test Split

```
In [7]: X = df_feat
        y = data['Class']

In [8]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

1.5 TensorFlow Model Creation & Training

```
In [9]: import tensorflow as tf

        df_feat.columns

Out[9]: Index(['Image.Var', 'Image.Skew', 'Image.Curt', 'Entropy'], dtype='object')

In [10]: # converting to numeric columns
         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')
         feat_cols = [image_var,image_skew,image_curt,entropy]

In [11]: classifier = tf.estimator.DNNClassifier(hidden_units=[10, 20, 10], n_classes=2,feature
         # creating a DNN classifier
         # has 2 classes and a [10,20,10] hidden unit layer structure

INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /var/folders/19/n10hsbyd7hg0srn_
INFO:tensorflow:Using config: {'_model_dir': '/var/folders/19/n10hsbyd7hg0srn_hx8kjl2m0000gp/T
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':


In [12]: input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,y=y_train,batch_size=20,shu
         # input function for the model, shuffle is enabled and batch size is set to 20

In [13]: classifier.train(input_fn=input_func,steps=500)
         # training classifier on 500 steps
```

```
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_core
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in eager and graph
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_esti
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_esti
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_core
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_esti
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_core
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_core
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
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.
WARNING:tensorflow:From /Users/kaushalrao/anaconda3/lib/python3.7/site-packages/tensorflow_core
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
INFO:tensorflow:Saving checkpoints for 0 into /var/folders/19/n10hsbyd7hg0srn_hx8kjl2m0000gp/T/
INFO:tensorflow:loss = 13.6387825, step = 1
INFO:tensorflow:Saving checkpoints for 48 into /var/folders/19/n10hsbyd7hg0srn_hx8kjl2m0000gp/T
INFO:tensorflow:Loss for final step: 0.25737643.
```

```
Out[13]: <tensorflow_estimator.python.estimator.canned.dnn.DNNClassifier at 0x1a3a93bd68>
```

1.6 Model Evaluation

```
In [14]: pred_fn = tf.estimator.inputs.pandas_input_fn(x=X_test,batch_size=len(X_test),shuffle=
         # another input function for predictions
```

```
In [15]: note_predictions = list(classifier.predict(input_fn=pred_fn))
         # predictions creation
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
```

```
INFO:tensorflow:Restoring parameters from /var/folders/19/n10hsbyd7hg0srn_hx8kjl2m0000gp/T/tmp
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

```
In [16]: final_preds  = []
         for pred in note_predictions:
             final_preds.append(pred['class_ids'][0])
```

```
In [17]: from sklearn.metrics import classification_report,confusion_matrix
         print(confusion_matrix(y_test,final_preds))
```

```
[[221   3]
 [  0 188]]
```

```
In [18]: print(classification_report(y_test,final_preds))
         # amazing classification performance!
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       224
           1       0.98      1.00      0.99       188

   micro avg       0.99      0.99      0.99       412
   macro avg       0.99      0.99      0.99       412
weighted avg       0.99      0.99      0.99       412
```

1.7 Quick Comparison to Random Forest

```
In [19]: from sklearn.ensemble import RandomForestClassifier
         rfc = RandomForestClassifier(n_estimators=200)
         rfc.fit(X_train,y_train)
         rfc_preds = rfc.predict(X_test)
```

```
In [20]: print(classification_report(y_test,rfc_preds))
         # RF performed really well also
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       224
           1       0.99      1.00      0.99       188

   micro avg       1.00      1.00      1.00       412
   macro avg       0.99      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412
```