In [1]:
```
#meta 1/25/2021 Poker Example 1
#src https://medium.com/@virgoady7/poker-hand-prediction-7a801e254acd
#Claim: Keras nn models predicts much higher than LogR, CART or SVM

#history
#1/25/202 ORIGINAL EXAMPLE + MY CODE DELTA
#      Original code errored out in Keras NN section: problem is with your lab
el-data shape
#      Fixed with $mycodedelta

#here 1/25/202 MANAGE DATA DOWNLOAD
#      Check if data already exists and downloaded if it doesn't
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from os import path
import warnings
warnings.filterwarnings('ignore')

#modeling
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

#from joblib import load, dump
```

Poker Example with Keras

0. Load Data

In [3]:
```
#$mycodedelta #was
#!wget http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-ha
nd-testing.data
#!wget http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-ha
nd-training-true.data
#!wget http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-ha
nd.names
```

In [4]:
```
#$mycodedelta
#check if data already downloaded
if path.exists('data/poker-hand.names'):
    print('Poker data already exists')
else:
    !wget http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker
-hand-testing.data -O 'data/poker-hand-testing.data'
    !wget http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker
-hand-training-true.data -O 'data/poker-hand-training-true.data'
    !wget http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker
-hand.names -O 'data/poker-hand.names'
```

```
Poker data already exists
```

1. Prep Data

note: When switch between train and test. SVM runs way longer. NN results are slightly better.
In reality need `train` , `valid` and `test` datsets.

In [5]:
```
data_test=pd.read_csv("data/poker-hand-training-true.data",header=None)
data_train = pd.read_csv("data/poker-hand-testing.data",header=None)
col=['Suit of card #1','Rank of card #1','Suit of card #2','Rank of card #2',
'Suit of card #3','Rank of card #3','Suit of card #4','Rank of card #4','Suit
 of card #5','Rank of card 5','Poker Hand']
col
```

Out[5]:
```
['Suit of card #1',
 'Rank of card #1',
 'Suit of card #2',
 'Rank of card #2',
 'Suit of card #3',
 'Rank of card #3',
 'Suit of card #4',
 'Rank of card #4',
 'Suit of card #5',
 'Rank of card 5',
 'Poker Hand']
```

In [6]:
```
data_train.columns=col
data_test.columns=col
```

In [7]:
```
y_train=data_train['Poker Hand']
y_test=data_test['Poker Hand']
y_train=pd.get_dummies(y_train)
y_test=pd.get_dummies(y_test)
```

In [8]:
```
x_train=data_train.drop('Poker Hand',axis=1)
x_test=data_test.drop('Poker Hand',axis=1)
```

In [9]:
```
print('Shape of Training Set:',x_train.shape)
print('Shape of Testing Set:',x_test.shape)
```

```
Shape of Training Set: (1000000, 10)
Shape of Testing Set: (25010, 10)
```

In [10]:
```
x_train.head()
```

Out[10]:

	Suit of card #1	Rank of card #1	Suit of card #2	Rank of card #2	Suit of card #3	Rank of card #3	Suit of card #4	Rank of card #4	Suit of card #5	Rank of card 5
0	1	1	1	13	2	4	2	3	1	12
1	3	12	3	2	3	11	4	5	2	5
2	1	9	4	6	1	4	3	2	3	9
3	1	4	3	13	2	13	2	1	3	6
4	3	10	2	7	1	2	2	11	4	9

In [11]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y_train=le.fit_transform(data_train['Poker Hand'])
y_test=le.transform(data_test['Poker Hand'])
```

In [12]:
```
y_train.shape, y_test.shape
```

Out[12]:
```
((1000000,), (25010,))
```

2. Model

Logistic Regression

In [13]:
```
clf = LogisticRegression(random_state=0, solver='lbfgs',max_iter=100,multi_cla
ss='ovr').fit(x_train, y_train)

#predict
y_hat=clf.predict(x_test)
accuracy_score(y_hat,y_test)
```

Out[13]:
```
0.4995201919232307
```

In [14]:
```
unique, counts = np.unique(y_hat, return_counts=True)
print (np.asarray((unique, counts)).T)
```

```
[[    0 25010]]
```

In [15]:
```
cm = confusion_matrix(y_test, y_hat)
print("Confusion matrix:\n{}".format(cm))
```

```
Confusion matrix:
[[12493     0     0     0     0     0     0     0     0     0]
 [10599     0     0     0     0     0     0     0     0     0]
 [ 1206     0     0     0     0     0     0     0     0     0]
 [  513     0     0     0     0     0     0     0     0     0]
 [   93     0     0     0     0     0     0     0     0     0]
 [   54     0     0     0     0     0     0     0     0     0]
 [   36     0     0     0     0     0     0     0     0     0]
 [    6     0     0     0     0     0     0     0     0     0]
 [    5     0     0     0     0     0     0     0     0     0]
 [    5     0     0     0     0     0     0     0     0     0]]
```

CART

Classification and Regression Trees or CART for short

In [16]:
```
decision_tree = DecisionTreeClassifier(random_state=0,max_depth = 3) #max_dept
h =2
decision_tree = decision_tree.fit(x_train,y_train)

#predict
y_hat = decision_tree.predict(x_test)
accuracy_score(y_hat,y_test)
```

Out[16]:
```
0.5038384646141544
```

In [17]:
```
unique, counts = np.unique(y_hat, return_counts=True)
print (np.asarray((unique, counts)).T)
```

```
[[    0 16313]
 [    1  8697]]
```

In [18]:
```
cm = confusion_matrix(y_test, y_hat)
print("Confusion matrix:\n{}".format(cm))
```

```
Confusion matrix:
[[8648 3845    0    0    0    0    0    0    0    0]
 [6646 3953    0    0    0    0    0    0    0    0]
 [ 683  523    0    0    0    0    0    0    0    0]
 [ 244  269    0    0    0    0    0    0    0    0]
 [  34   59    0    0    0    0    0    0    0    0]
 [  34   20    0    0    0    0    0    0    0    0]
 [  14   22    0    0    0    0    0    0    0    0]
 [   2    4    0    0    0    0    0    0    0    0]
 [   3    2    0    0    0    0    0    0    0    0]
 [   5    0    0    0    0    0    0    0    0    0]]
```

SVM

We plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate

In [19]:
```
clf = svm.LinearSVC()
clf.fit(x_train,y_train)

#predict
y_hat = clf.predict(x_test)
accuracy_score(y_hat,y_test)
```

Out[19]:
```
0.44530187924830067
```

In [20]:
```
unique, counts = np.unique(y_hat, return_counts=True)
print (np.asarray((unique, counts)).T)
```

```
[[    0  3361]
 [    1 21649]]
```

In [21]:
```
cm = confusion_matrix(y_test, y_hat)
print("Confusion matrix:\n{}".format(cm))
```

```
Confusion matrix:
[[ 1843 10650     0     0     0     0     0     0     0     0]
 [ 1305  9294     0     0     0     0     0     0     0     0]
 [  138  1068     0     0     0     0     0     0     0     0]
 [   60   453     0     0     0     0     0     0     0     0]
 [    4    89     0     0     0     0     0     0     0     0]
 [    7    47     0     0     0     0     0     0     0     0]
 [    3    33     0     0     0     0     0     0     0     0]
 [    0     6     0     0     0     0     0     0     0     0]
 [    0     5     0     0     0     0     0     0     0     0]
 [    1     4     0     0     0     0     0     0     0     0]]
```

Neural Net with Keras

A neural network is a progression of algorithms that attempts to perceive fundamental connections in a lot of information through a procedure that copies the manner in which the human brain works. Neural network can adjust to changing input; so the network produces the most ideal outcome without expecting to redesign the output criteria. To create NN we used Keras library which is a high-level API wrapper for the low-level API, capable of running on top of TensorFlow, CNTK, or Theano. My neural network architecture comprised of 3 dense layers with respectively 15,10 and 10 nodes in each layer.

In [22]:
```
#$mycodedelta
#code below errors out: ValueError: Error when checking target: expected dense
_3 to have shape (10,) but got array with shape (1,)
#src https://stackoverflow.com/questions/49392972/error-when-checking-target-e
xpected-dense-3-to-have-shape-3-but-got-array-wi/55992428
#issue: problem is with label-data shape
#      Keras expects y-data in (N, 10) shape, not (N,)
#      was: y_train.shape, y_test.shape
#      was: ((25010,), (1000000,))
#fix: Recode labels using to_categorical to get the correct shape of inputs
from keras.utils import to_categorical
y_train_nn = to_categorical(y_train)
y_test_nn = to_categorical(y_test)

y_train_nn.shape, y_test_nn.shape
```

```
Using TensorFlow backend.
```

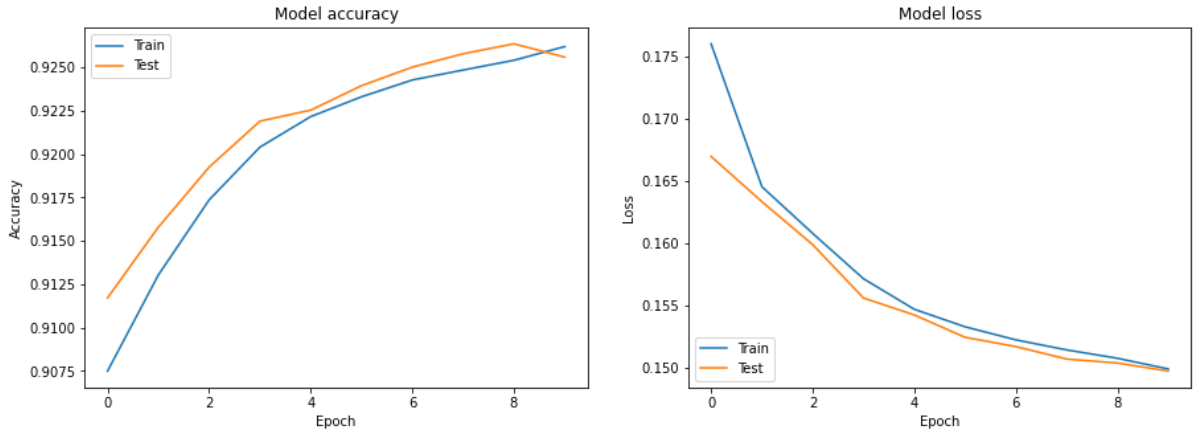
Out[22]:
```
((1000000, 10), (25010, 10))
```

In [23]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
from keras import regularizers


model = Sequential()
model.add(Dense(15, activation='relu', input_dim=10))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train_nn, epochs = 10, batch_size = 256, verbos
e=1,validation_data=(x_test,y_test_nn),shuffle=True) #$mycodedelta


score = model.evaluate(x_test, y_test_nn, batch_size=256) #$mycodedelta
```

```
WARNING:tensorflow:From D:\Anaconda3\envs\hack-keras\lib\site-packages\tensor
flow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (fro
m tensorflow.python.ops.array_ops) is deprecated and will be removed in a fut
ure version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From D:\Anaconda3\envs\hack-keras\lib\site-packages\keras
\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecate
d. Please use tf.compat.v1.global_variables instead.

Train on 1000000 samples, validate on 25010 samples
Epoch 1/10
1000000/1000000 [==============================] - 6s 6us/step - loss: 0.1760
- accuracy: 0.9075 - val_loss: 0.1670 - val_accuracy: 0.9117
Epoch 2/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1645
- accuracy: 0.9130 - val_loss: 0.1633 - val_accuracy: 0.9158
Epoch 3/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1608
- accuracy: 0.9174 - val_loss: 0.1599 - val_accuracy: 0.9192
Epoch 4/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1572
- accuracy: 0.9204 - val_loss: 0.1556 - val_accuracy: 0.9219
Epoch 5/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1547
- accuracy: 0.9222 - val_loss: 0.1543 - val_accuracy: 0.9225
Epoch 6/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1533
- accuracy: 0.9233 - val_loss: 0.1525 - val_accuracy: 0.9239
Epoch 7/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1522
- accuracy: 0.9243 - val_loss: 0.1517 - val_accuracy: 0.9250
Epoch 8/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1514
- accuracy: 0.9248 - val_loss: 0.1507 - val_accuracy: 0.9258
Epoch 9/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1508
- accuracy: 0.9254 - val_loss: 0.1504 - val_accuracy: 0.9263
Epoch 10/10
1000000/1000000 [==============================] - 5s 5us/step - loss: 0.1499
- accuracy: 0.9262 - val_loss: 0.1497 - val_accuracy: 0.9256
25010/25010 [==============================] - 0s 2us/step
```

In [24]:
```
plt.figure(figsize=(15, 5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy']) #$mycodedelta
plt.plot(history.history['val_accuracy'])#$mycodedelta
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower left')
```

Out[24]: <matplotlib.legend.Legend at 0x22c09e29108>



Evaludate NN model

by predicting on the given test set (which unfortunately has been used for validation, too).

In [25]:
```
#predict
y_hat = model.predict(x_test) #numpy.ndarray
y_hat_class = y_hat.argmax(axis=1)
print(y_hat.shape, y_hat_class.shape)
accuracy_score(y_hat_class, y_test)
```

```
(25010, 10) (25010,)
```

Out[25]: 0.624750099960016

In [26]:
```
unique, counts = np.unique(y_hat_class, return_counts=True)
print (np.asarray((unique, counts)).T)
```

```
[[    0 15773]
 [    1  9178]
 [    2     4]
 [    3    55]]
```

In [27]:
```
cm = confusion_matrix(y_test, y_hat_class)
print("Confusion matrix:\n{}".format(cm))
```

```
Confusion matrix:
[[10146  2347     0     0     0     0     0     0     0     0]
 [ 5146  5451     0     2     0     0     0     0     0     0]
 [  350   835     1    20     0     0     0     0     0     0]
 [   68   415     3    27     0     0     0     0     0     0]
 [   10    83     0     0     0     0     0     0     0     0]
 [   44    10     0     0     0     0     0     0     0     0]
 [    5    26     0     5     0     0     0     0     0     0]
 [    0     5     0     1     0     0     0     0     0     0]
 [    0     5     0     0     0     0     0     0     0     0]
 [    4     1     0     0     0     0     0     0     0     0]]
```

Summary

The author claims that the Neural Network using Keras Library enables us to produce the most accurate results above all. I further evaluated the model results by predicting on the `test` ds and found that NN predictions didn't perform anywhere close to 90% and more like other models ~ 50%.

In ML, using the same `test` ds for validation and testing is not a valid technique. Next step should be to truly have `train`, `validation` and `test` sets and see how all the models fair with a holdout dataset.

Src: https://keras.io/guides/training_with_built_in_methods/ (https://keras.io/guides/training_with_built_in_methods/) Here's what the typical end-to-end workflow looks like, consisting of:

- Training
- Validation on a holdout set generated from the original training data
- Evaluation on the test data

In [28]:
```
mystop
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-255d8eb4da7e> in <module>
----> 1 mystop

NameError: name 'mystop' is not defined
```

Xtra

In [ ]:
```
#$xtra my export data for reuse
dump(data_train, 'data/poker_ex1_data_train.pkl')
dump(data_test, 'data/poker_ex1_data_test.pkl')
```