Poker Predictions

Austin DeVore
Dan Kramer
Devan Cherne
Ezrelle Myhre-Hager

What's the Problem?

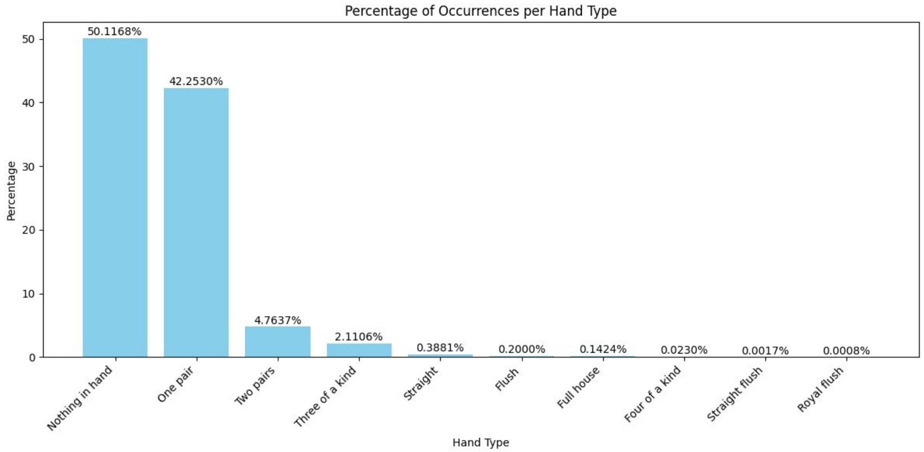
- When given a hand of 5 cards, can the model accurately predict the best playable hand?
- Possible Hands: No hand, Pair, Two Pair, Three of a Kind, Straight, Flush, Full House, Four of a Kind, Straight Flush, Royal Flush
- How do different types of ML models compare when making these predictions?
- NOT predicting the next 5 cards from the deck!

SQL and Data Analysis

- Exploratory Analysis
- Over 1m rows of data (“hands”)
- Example: Counts of each hand type in dataset

```
# Data exploration through queries...


start_time = time.time()

spark.sql("""

SELECT Class, COUNT(Class) AS Occurrences
FROM hands
GROUP BY Class
ORDER BY Occurrences DESC
""").show()

print("--- %s seconds ---" % (time.time() - start_time))
```

```
+-----+-----------+
|Class|Occurrences|
+-----+-----------+
|    0|     513702|
|    1|     433097|
|    2|      48828|
|    3|      21634|
|    4|       3978|
|    5|       2050|
|    6|       1460|
|    7|        236|
|    8|         17|
|    9|          8|
+-----+-----------+

--- 3.259308099746704 seconds ---
```



Features and Classes

Additional Variable Information

1) S1 "Suit of card #1"
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}

2) C1 "Rank of card #1"
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)

3) S2 "Suit of card #2"
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}

4) C2 "Rank of card #2"
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)

5) S3 "Suit of card #3"
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}

6) C3 "Rank of card #3"
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)

7) S4 "Suit of card #4"
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}

8) C4 "Rank of card #4"
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)

9) S5 "Suit of card #5"
   Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}

10) C5 "Rank of card 5"
   Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)

11) CLASS "Poker Hand"
   Ordinal (0-9)

   0: Nothing in hand; not a recognized poker hand
   1: One pair; one pair of equal ranks within five cards
   2: Two pairs; two pairs of equal ranks within five cards
   3: Three of a kind; three equal ranks within five cards
   4: Straight; five cards, sequentially ranked with no gaps
   5: Flush; five cards with the same suit
   6: Full house; pair + different rank three of a kind
   7: Four of a kind; four equal ranks within five cards
   8: Straight flush; straight + flush
   9: Royal flush; {Ace, King, Queen, Jack, Ten} + flush

Logistic Regression

Description

This method is used to predict the categorical dependent variable. This meaning to predict a binary dependent variable thus making sense on why the score is .50117.

Accuracy

```
# Calc Accuracy Score
from sklearn.metrics import accuracy_score
# Display the accuracy score for the test dataset.
accuracy_score(y_test, predictions)
```

```
0.5011687668046814
```

Results

	Prediction	Actual
0	0	0
1	0	1
2	0	1
3	0	0
4	0	1
...	...	...
256248	0	0
256249	0	1
256250	0	2
256251	0	0
256252	0	0

256253 rows × 2 columns

Decision Tree

Description

Decision Tree is a hierarchical type of model. This can be used for non-linear relationships and able to handle numerical data, which is how our data is set up numbering the types of hands by numbers.

Confusion Matrix

Confusion Matrix

	Nothing in Hand	One Pair	Two Pairs	Three of a kind	Straight	Flush	Full house	Four of a kind	Straight flush	Royal flush
Nothing in hand	90758	34575	1825	586	112	505	16	1	1	1
One Pair	32256	64784	7476	2774	605	113	124	7	4	0
Two Pairs	1451	6358	3884	409	65	5	115	18	1	0
Three of a kind	415	2447	345	2097	32	0	116	18	0	0
Straight	110	523	75	39	288	0	4	0	1	0
Flush	300	81	1	0	1	103	0	0	0	3
Full house	7	93	103	95	0	0	53	2	0	0
Four of a kind	1	4	18	26	0	0	3	13	0	0
Straight flush	0	2	0	0	3	1	0	0	0	0
Royal flush	0	0	0	0	0	1	0	0	0	0

Results

```
Accuracy Score : 0.6321096728623665
Classification Report
              precision    recall  f1-score   support

           0       0.72      0.71      0.72    128380
           1       0.60      0.60      0.60    108143
           2       0.28      0.32      0.30     12306
           3       0.35      0.38      0.36      5470
           4       0.26      0.28      0.27      1040
           5       0.14      0.21      0.17       489
           6       0.12      0.15      0.14       353
           7       0.22      0.20      0.21        65
           8       0.00      0.00      0.00         6
           9       0.00      0.00      0.00         1

    accuracy                           0.63    256253
   macro avg       0.27      0.28      0.28    256253
weighted avg       0.64      0.63      0.63    256253
```

Linear SVM

- What is Linear SVM?
  - Linear Support Vector is a supervised learning algorithm used for classification tasks. It finds the best line that separates the data points of different classes with the max margin.
- Why would you use Linear SVM?
  - Multi-dimensional dataset
  - Complex Patterns
- Results:
  - Accuracy: 49.96%
  - Confusion Matrix (next slide)

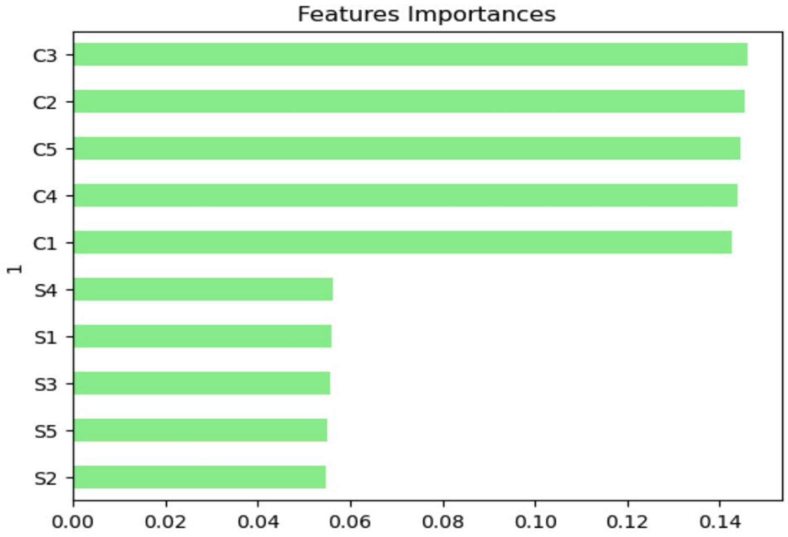
Linear SVM Confusion Matrix

Confusion Matrix

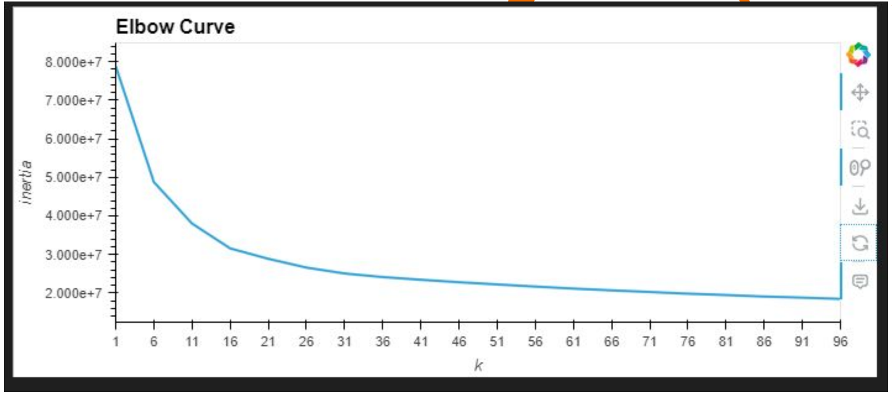
	Predicted No Hand	Predicted Pair	Predicted Two Pair	Predicted Three of a Kind	Predicted Straight	Predicted Flush	Predicted Full House	Predicted Four of a Kind	Predicted Straight Flush	Predicted Royal Flush
Actual No Hand	102428	0	0	0	0	0	0	0	0	0
Actual Pair	86945	0	0	0	0	0	0	0	0	0
Actual Two Pair	9691	0	0	0	0	0	0	0	0	0
Actual Three of a Kind	4352	0	0	0	0	0	0	0	0	0
Actual Straight	808	0	0	0	0	0	0	0	0	0
Actual Flush	405	0	0	0	0	0	0	0	0	0
Actual Full House	308	0	0	0	0	0	0	0	0	0
Actual Four of a Kind	60	0	0	0	0	0	0	0	0	0
Actual Straight Flush	3	0	0	0	0	0	0	0	0	0
Actual Royal Flush	2	0	0	0	0	0	0	0	0	0

Random Forest

- What is Random Forest?
  - Random Forest is a machine learning model that constructs multiple decision trees during training then outputs the class based on the classification of the individual trees
- Why would you use it?
  - Improves accuracy by reducing overfitting
  - Useful with higher dimensional datasets
- Results:
  - Accuracy: 76.06%
- Optimization Results:
  - GridSearchSV (tuning estimators)
  - Failed because of limited resources



K-Nearest Neighbors (KNN)




```
/Users/Devan_user_friendly/anaconda3/envs/dev/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1469: Undefined
  _warn_prf(average, modifier, msg_start, len(result))
/Users/Devan_user_friendly/anaconda3/envs/dev/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1469: Undefined
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.75      0.60      0.67    162300
           1       0.46      0.53      0.49     93557
           2       0.01      0.32      0.01       220
           3       0.00      0.16      0.00        31
           4       0.00      0.00      0.00         1
           5       0.29      0.99      0.45       144
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0

    accuracy                           0.57    256253
   macro avg       0.15      0.26      0.16    256253
weighted avg       0.64      0.57      0.60    256253

/Users/Devan_user_friendly/anaconda3/envs/dev/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1469: Undefined
  _warn_prf(average, modifier, msg_start, len(result))
```

- What is KNN and why use it?
- Utilized Ravel so that the model could read the data correctly.
- Results: Best k values are k=6 and k=16.
- k=16 produced an accuracy of 57%.
    - k=6 wasn’t that off.

Deep Neural Network

```python
# Create a method that creates a new Sequential model with hyperparameter options
def create_model(hp):
  nn_model = tf.keras.models.Sequential()
    # Allow kerastuner to decide which activation function to use in hidden layers
  activation = hp.Choice('activation', ['relu', 'tanh'])
    # Allow kerastuner to decide number of neurons in first layer
  nn_model.add(tf.keras.layers.Dense(units=hp.Int('first_units',
      min_value=1 , max_value=50, step=10), activation=activation, input_dim=10))
    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
  for i in range(hp.Int('num_layers', 1, 5)):
    nn_model.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
    min_value=1,
    max_value=50,
    step=10),
    activation=activation))

  nn_model.add(tf.keras.layers.Dense(units=10, activation='softmax'))
    # Compile the model
  nn_model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

  return nn_model
```

```python
# Import the kerastuner library
import keras_tuner as kt
# Create a `Hyperband()` tuner instance
tuner = kt.Hyperband(
    create_model,
    objective="val_accuracy",
    max_epochs=10,
    hyperband_iterations=1
)
```

```python
# Run the kerastuner search for best hyperparameters
tuner.search(X_train_scaled, y_train, epochs=10, validation_data=(X_test_scaled, y_test))
```

```
Trial 30 Complete [00h 09m 04s]
val_accuracy: 0.5233577489852905

Best val_accuracy So Far: 0.9980565905570984
Total elapsed time: 02h 08m 43s
```

```python
# Get top 3 model hyperparameters and print the values
best_hyper = tuner.get_best_hyperparameters(3)
for param in best_hyper:
  print(param.values)
```

```
{'activation': 'relu', 'first_units': 21, 'num_layers': 2, 'units_0': 41, 'units_1': 21, 'units_2': 11, 'units_3': 1, 'units_4': 21, 'tuner/epochs': 10, 'tuner/initial_epoch': 4,
{'activation': 'relu', 'first_units': 11, 'num_layers': 5, 'units_0': 41, 'units_1': 21, 'units_2': 1, 'units_3': 31, 'units_4': 21, 'tuner/epochs': 10, 'tuner/initial_epoch': 4,
{'activation': 'relu', 'first_units': 21, 'num_layers': 2, 'units_0': 41, 'units_1': 21, 'units_2': 11, 'units_3': 1, 'units_4': 21, 'tuner/epochs': 4, 'tuner/initial_epoch': 2, '
```

- Utilize the Keras Tuner for help determining parameters

- Best model characteristics:
  - Activation Function: Relu
  - Initial Layer: 21 neurons
  - 1st Hidden Layer: 41 neurons
  - 2nd Hidden Layer: 21 neurons

- Adjustments required to account for multiple classifications
  - Output Layer: 10 neurons, SoftMax Activation Function

The Results

1st - Deep Learning: 99.81%

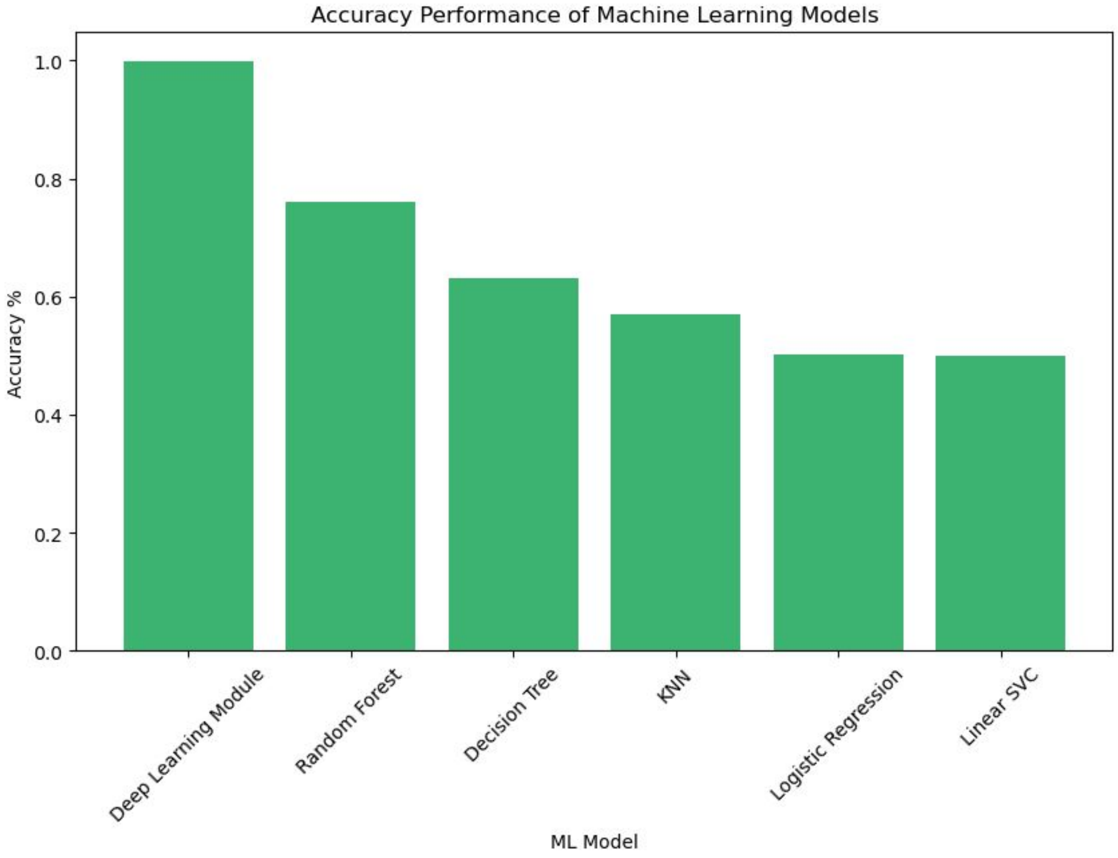
2nd - Random Forest: 76.06%

3rd - Decision Tree: 63.21%

4th - KNN: 57.00%

5th - Logistic Regression: 50.12%

6th - Linear SVC: 49.96%



Further Steps?

After testing the models, it is clear that from these choices, the Deep Neural Network would be the **best** approach to take when trying to solve problems of this type. That aside, additional time may have allowed the following explorations that could have improved some of the model's accuracy:

- Tuning Random Forest Estimators
- Tuning Iterations in Logistic Regression

Other powerful models could also be explored but may be unnecessary due to the success of the DNN model.

References

- Cattral, Robert and Oppacher, Franz. (2007). Poker Hand. UCI Machine Learning Repository. https://doi.org/10.24432/C5KW38.