Beating Vegas Using Machine Learning

By: Michael, Dylan, and Shawn

Overarching Questions and Motivation

How can we use Machine Learning to predict the outcomes for games in the NFL and help place bets on teams?

- Increased popularity in sports gambling
  - 13 states have legal, online sports gambling
  - Large state tax revenue (source)
    - NJ -> $61,484,646 (since June 2018)
      - amount wagered -> $6,865,133,129

- As popularity rises, more available data.

- More data → better chance of finding trends → more money to be made!

Possible ML Outcomes per Game - Optimizing the Tool

	Home Team	Away Team	
Recommended not to bet →	0	0	
	0	1	← Bet on Away
Bet on Home →	1	0	
	1	1	← Recommended not to bet

The model is forced to pick all games, but you can likely improve your outcome by sticking with picks where the model agrees with itself.

The Data

The data we chose to predict the winner or loser includes 6 dimensions:

- Home vs. Away
- Net yards
- Net points
- Net turnovers
- Weekly Power Ranking
- Point spread

We are looking at **2,304 games:**

- **4,608 rows (Home/Away)**
  - **Win**
    - **2,300 Wins**
    - **2,308 Loses**
  - **WinSpread**
    - **2,370 Wins**
    - **2,238 Loses**

Win vs WinSpread

For each model, we looked at both:

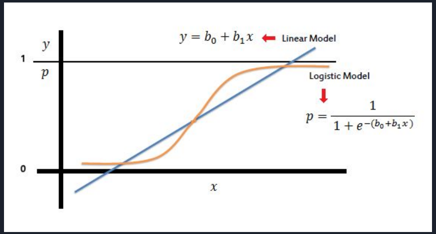
- Win
  - The model simply tries to predict whether the team presented is the loser or winner (0 or 1).

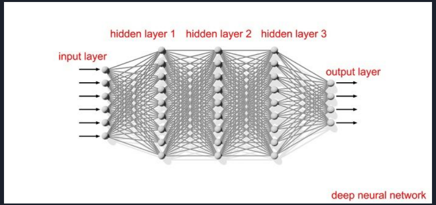
- WinSpread
  - The model tries to predict whether the team presented is the loser or winner (0 or 1) based on the spread.
  - The spread acts as a handicap - one team (the favorite) has to give points to the other team (the underdog).
    - Ex: Team A is favored to win by 5.5 points
      - If Team A wins by 6, they win versus the spread.
      - If Team A wins by 5 or less, they lose vs the spread.
      - If Team A loses, they lose vs the spread.

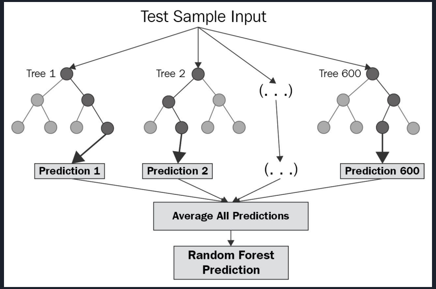
Models

We decided to use:

1. Logistic Regression
2. Deep Learning
3. Random Forest







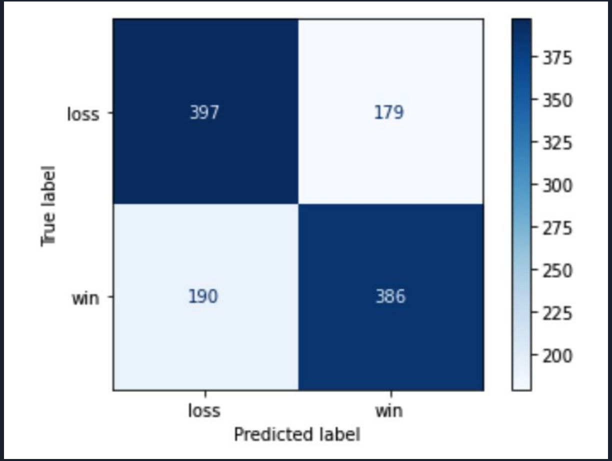
Logistic Regression: Win

Training set accuracy: 0.663
Test set accuracy: 0.672

Mean Squared Error: 0.33
Area Under the Curve: 0.67

First 10 Predictions: [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]

First 10 Actual labels: [1, 0, 0, 1, 0, 0, 0, 1, 1, 1]



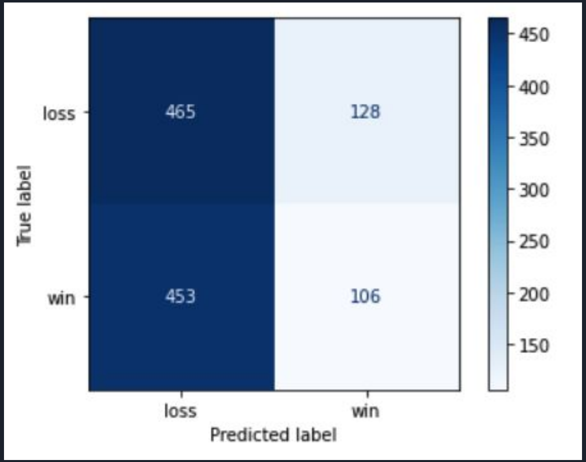
Logistic Regression: WinSpread

Training set accuracy: 0.511
Test set accuracy: 0.502

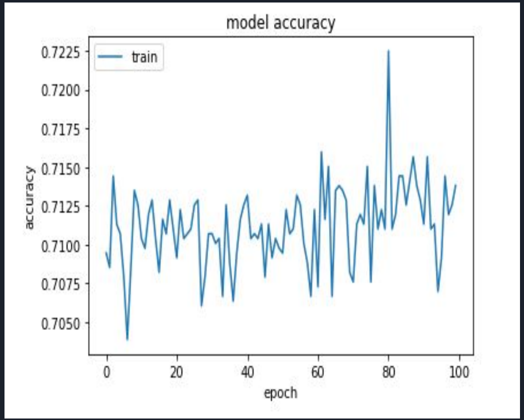
Mean Squared Error: 0.46
Area Under the Curve: 0.49

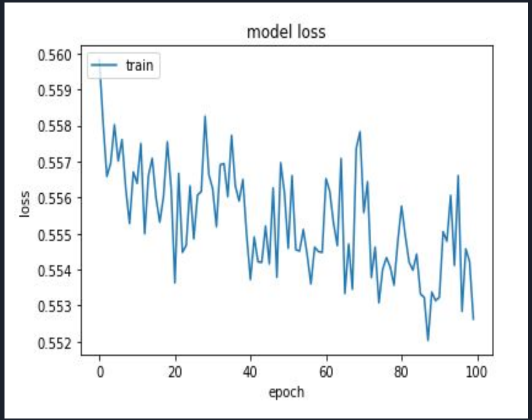
First 10 Predictions: [0, 0, 0, 0, 0, 0, 1, 0, 0, 1]

First 10 Actual labels: [0, 0, 0, 1, 0, 0, 0, 1, 1, 1]



Visualization for Deep Learning Model - Win

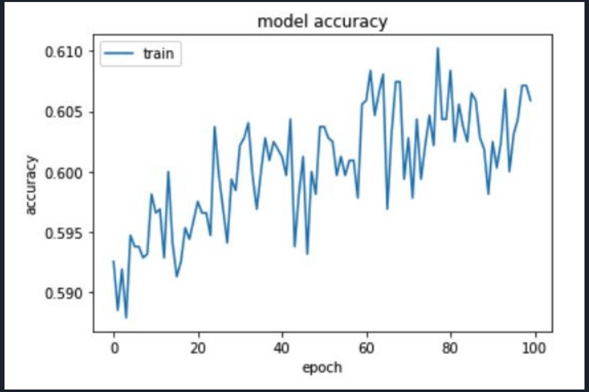


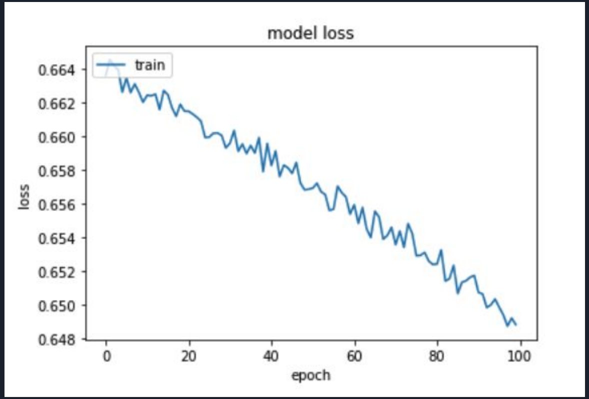


Loss: 0.60

Accuracy: 0.67

Visualization for Deep Learning Model - WinSpread





Loss: 0.71

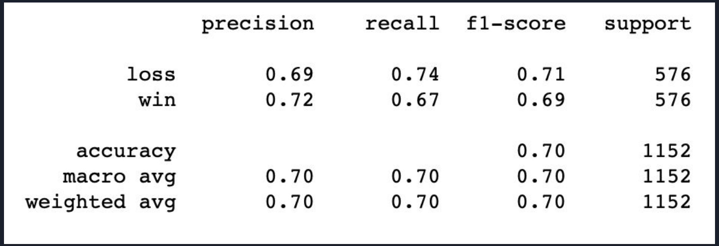
Accuracy: 0.53

Random Forest for Win

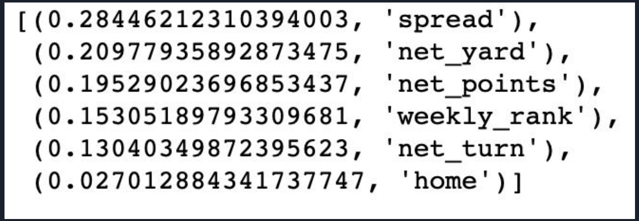
Used GridSearch → 'criterion': 'entropy', 'n_estimators': 400

Accuracy: 70%

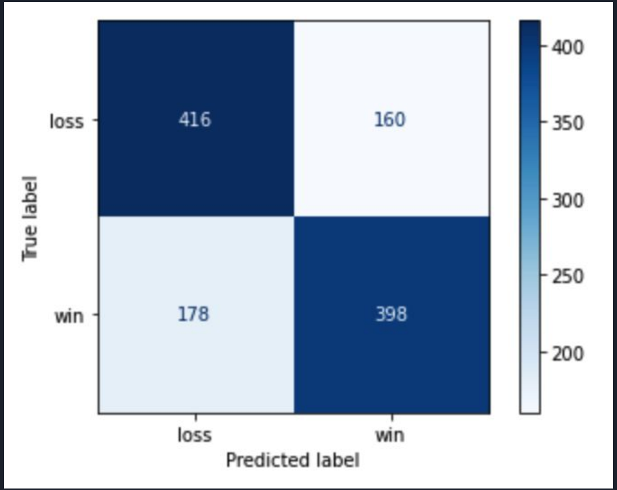
Area Under the Curve: 0.70



```
              precision    recall  f1-score   support

        loss       0.69      0.74      0.71       576
         win       0.72      0.67      0.69       576

    accuracy                           0.70      1152
   macro avg       0.70      0.70      0.70      1152
weighted avg       0.70      0.70      0.70      1152
```



```
[(0.28446212310394003, 'spread'),
 (0.20977935892873475, 'net_yard'),
 (0.19529023696853437, 'net_points'),
 (0.15305189793309681, 'weekly_rank'),
 (0.13040349872395623, 'net_turn'),
 (0.027012884341737747, 'home')]
```

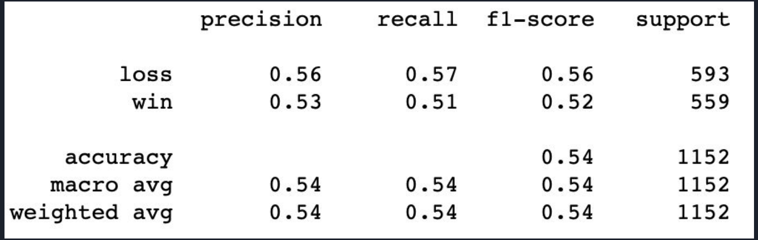


Random Forest for WinSpread

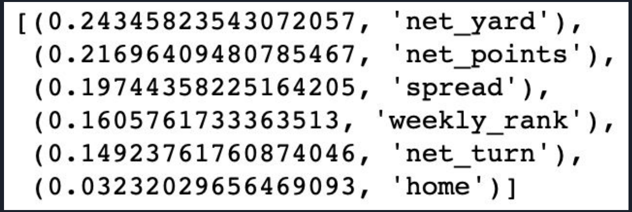
Used GridSearch → 'criterion': 'entropy', 'n_estimators': 400

Accuracy: 54.4%

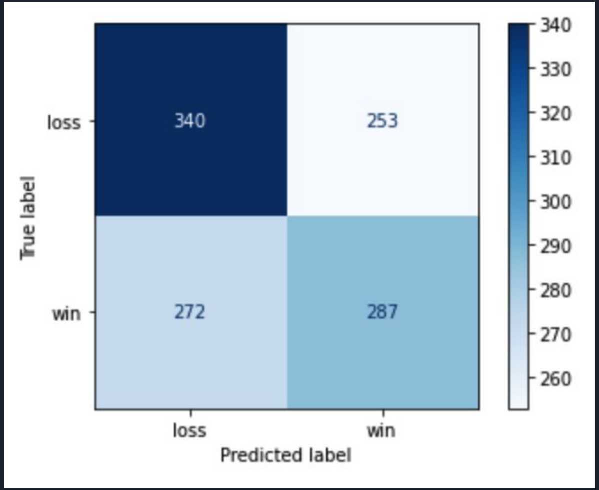
Area Under the Curve: 0.543



```
              precision    recall  f1-score   support

        loss       0.56      0.57      0.56       593
         win       0.53      0.51      0.52       559

    accuracy                           0.54      1152
   macro avg       0.54      0.54      0.54      1152
weighted avg       0.54      0.54      0.54      1152
```



```
[(0.24345823543072057, 'net_yard'),
 (0.21696409480785467, 'net_points'),
 (0.19744358225164205, 'spread'),
 (0.1605761733363513, 'weekly_rank'),
 (0.14923761760874046, 'net_turn'),
 (0.0323202965646909З, 'home')]
```



Why is 54.4% good?

Odds are placed with vigorish (also known as juice, the cut, the take, the margin, the house edge or simply the vig).

When Vegas creates a spread, the goal is to make the game 50-50. However, the odds are not even. Instead, they are placed somewhere around -110 (meaning you need to risk $110 to win $100). This extra $10 is the vig and is where Vegas makes all it’s money. (Think of the zeros/green on a roulette wheel.)

By using an expected outcome formula, we find that being able to correctly pick the spread at >52% means you can “beat the odds” on your typical -110 bet and win money.

$$\mathrm{E}[X] = \sum_{i=1}^{k} x_i\, p_i = x_1 p_1 + x_2 p_2 + \cdots + x_k p_k.$$

W = probability of winning the bet
E = expected payout

E = \$100(0.544) + (-\$110)(1 - 0.544)
E = \$54.4 - \$110(0.456)
E = \$54.4 - \$50.16
E = \$4.24

0 < \$100(W) + (-\$110)(1 - W)
0 < \$100(W) + \$110(W) - \$110
\$110 < \$210(W)
\$110/\$210 < W
52% < W

Appendix

Data retrieval

We were unable to find one premade csv with everything, so we needed to find the data from multiple sources and join them.

Using the tools we learned in class(Pandas - read_html), we scraped and looped through different tables on websites.

Then, we used a combination of loops, conditionals, and inline calculations to merge/join the different tables to create a final dataframe packaging all the data.

Sources:

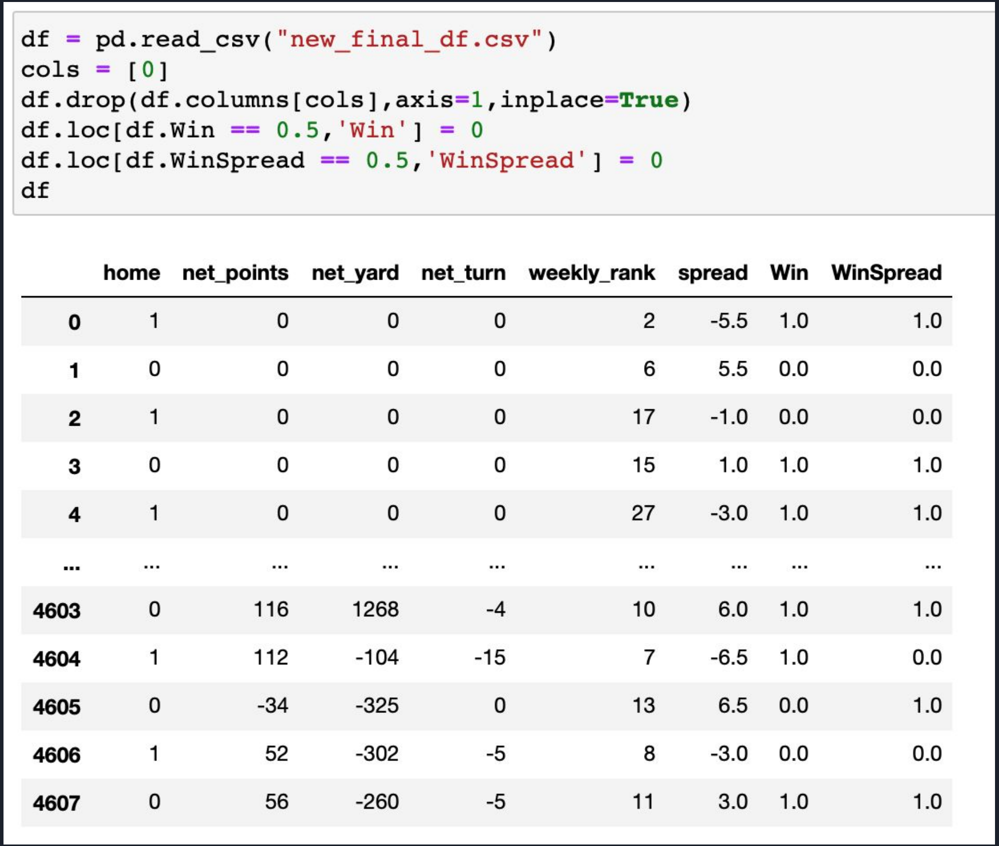
https://www.pro-football-reference.com/years/2007/games.htm
http://www.espn.com/nfl/powerrankings/_/year/2010/week/2
https://www.kaggle.com/tobycrabtree/nfl-scores-and-betting-data/home?select=spreadspoke_scores.csv

Exploring the data / Cleaning the data

Some of the data cleaning that we had to do:

- Calculate the net yards, net turnovers, and net points each team (home vs away) had before the given game.
- Merge the teams’ weekly ranking for the game, so we can see how each team compared.
- Our data also included ties which we found to cause some trouble.
    - We decided to change a draw to a loss for both teams as the bet would not have a winner.
    - This allowed us to create a simple classification model - two outcomes, winner or loser.
- We also had some data that was missing scraping from some website.
    - We had to manually find the rows (4 total) that were missing data and insert it.

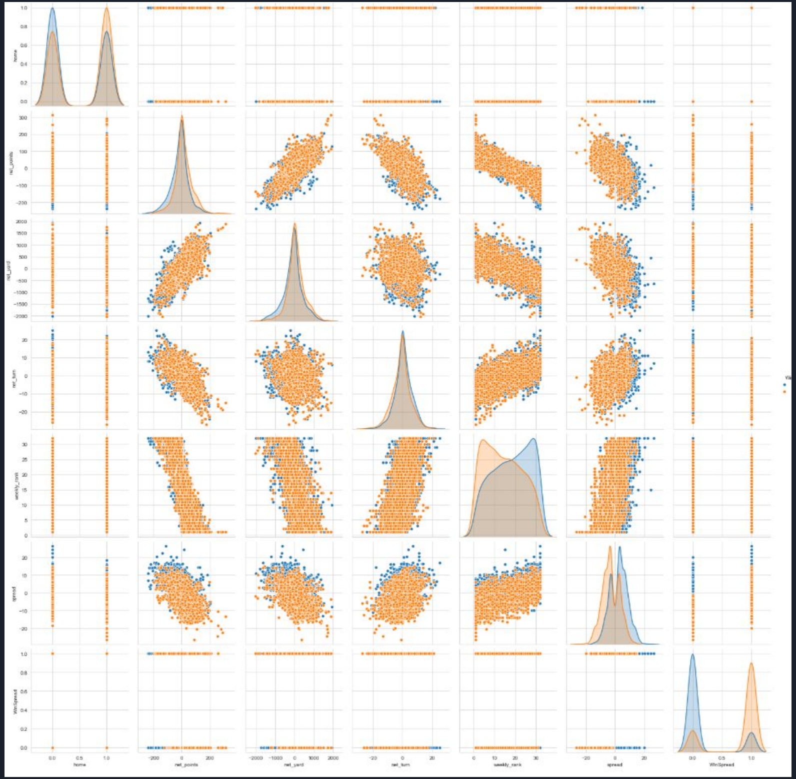
Final Data Set



```
df = pd.read_csv("new_final_df.csv")
cols = [0]
df.drop(df.columns[cols],axis=1,inplace=True)
df.loc[df.Win == 0.5,'Win'] = 0
df.loc[df.WinSpread == 0.5,'WinSpread'] = 0
df
```

	home	net_points	net_yard	net_turn	weekly_rank	spread	Win	WinSpread
0	1	0	0	0	2	-5.5	1.0	1.0
1	0	0	0	0	6	5.5	0.0	0.0
2	1	0	0	0	17	-1.0	0.0	0.0
3	0	0	0	0	15	1.0	1.0	1.0
4	1	0	0	0	27	-3.0	1.0	1.0
...	...	...	...	...	...	...	...	...
4603	0	116	1268	-4	10	6.0	1.0	1.0
4604	1	112	-104	-15	7	-6.5	1.0	0.0
4605	0	-34	-325	0	13	6.5	0.0	1.0
4606	1	52	-302	-5	8	-3.0	0.0	0.0
4607	0	56	-260	-5	11	3.0	1.0	1.0

Initial Visualization



We used seaborn to plot all of the features we have against each other.

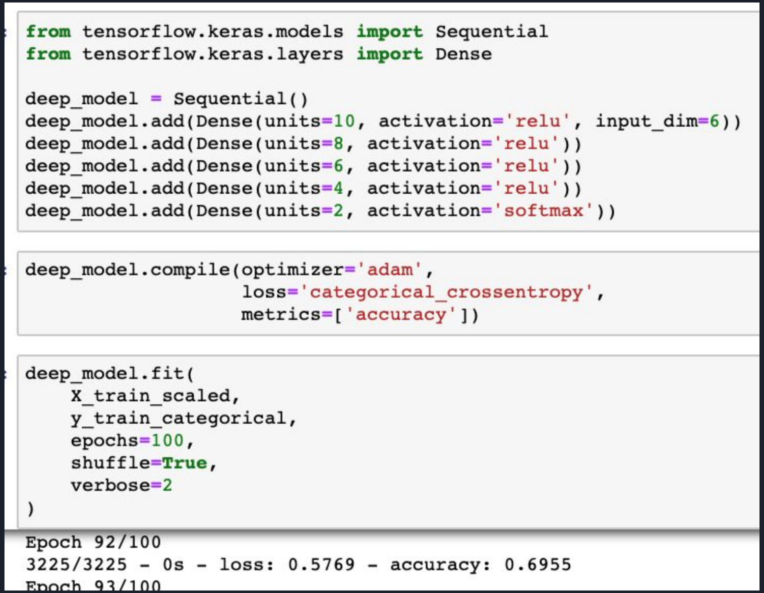
We used the hue as either being a win or a loss.

This shows that weekly rank and spread should be our best predictors of winning or losing.

Deep Learning Model

Using tensorflow that we learned in class we made a deep learning model with:

- 3 hidden layers
- A scaled X;
- the optimizer as ‘adam’;
- loss as ‘categorical cross entropy’;
- the metric being accuracy;
- epoch of 100;
- shuffle as TRUE; and
- verbose of 2.



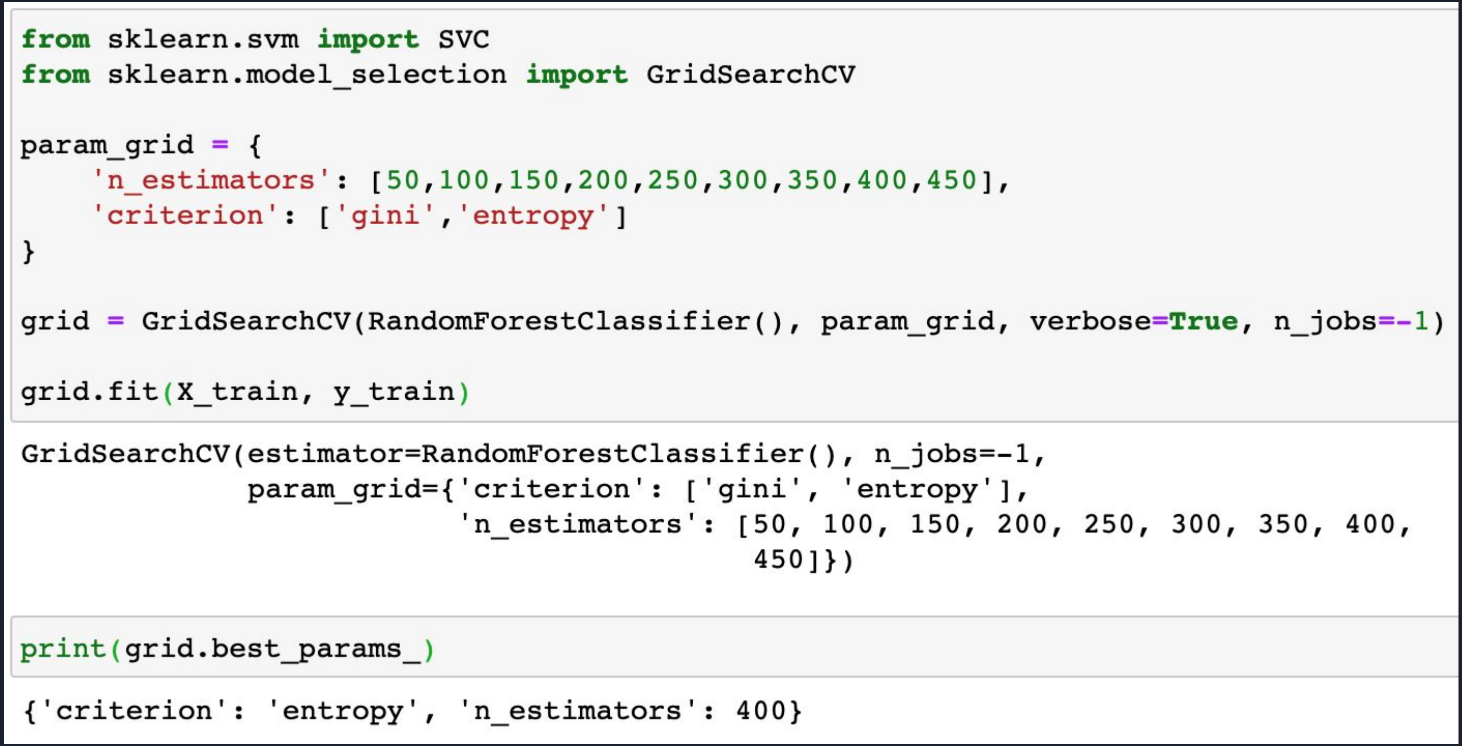
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

deep_model = Sequential()
deep_model.add(Dense(units=10, activation='relu', input_dim=6))
deep_model.add(Dense(units=8, activation='relu'))
deep_model.add(Dense(units=6, activation='relu'))
deep_model.add(Dense(units=4, activation='relu'))
deep_model.add(Dense(units=2, activation='softmax'))
```

```
deep_model.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

```
deep_model.fit(
    X_train_scaled,
    y_train_categorical,
    epochs=100,
    shuffle=True,
    verbose=2
)
```

```
Epoch 92/100
3225/3225 - 0s - loss: 0.5769 - accuracy: 0.6955
Epoch 93/100
```

GridSearch for Random Forest



```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50,100,150,200,250,300,350,400,450],
    'criterion': ['gini','entropy']
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, verbose=True, n_jobs=-1)

grid.fit(X_train, y_train)
```

```
GridSearchCV(estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'n_estimators': [50, 100, 150, 200, 250, 300, 350, 400,
                                          450]})
```

```
print(grid.best_params_)
```

```
{'criterion': 'entropy', 'n_estimators': 400}
```