T20- CRICKET PREDICTOR

GROUP -2 (Zach Buckley and Vibhu Pillai)

PROBLEM STATEMENT

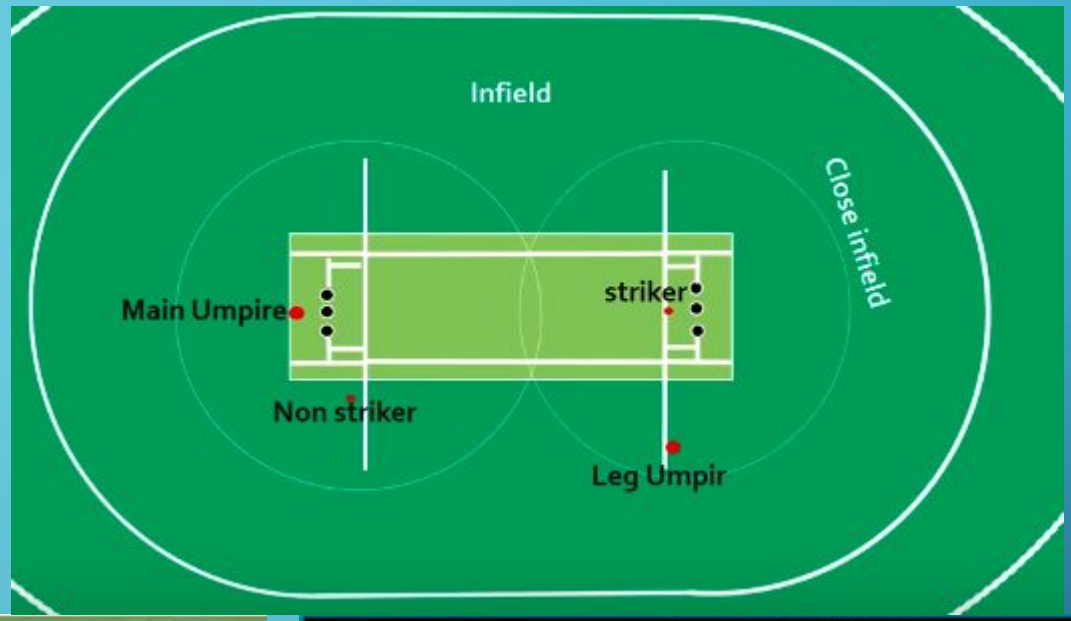
Predicting a cricket game is a challenge considering the very many factors that affect the outcome of the game.

Can we predict the outcome of a T20 cricket match after the 1st inning has been completed?

WHY AND WHAT IS CRICKET

Cricket is a highly technical sport with a lot of factors that influence the outcome of the game. Scores differ based on venues. The pitch on which the game is played determines the output. For example a dry wicket would help the ball to come quicker to the batsman while it would assist the slower bowlers. The game is divided into 2 innings, one for each team. The team batting first sets a score with the stipulated 120 pitches. The team batting 2nd now has score one more than the runs scored by the 1st team. For our project we have tried to predict the outcome of the game based on the first innings score as the input.

MORE ON CRICKET



SCL 2K19	Quarter Final Match-2		SCL 2K19
Customize this design with your photos and text			
Gridz		6 overs	51-3
Arun	16 (11)	Shamnad	1-8
Ivin	12 (11)	Gokul	1-13
Krishnanunni	7 (9)	Steny	0-15
Team Budweiser		6 overs	22-6
Bazi	8 (15)	Rishin	2-2
Gokul	4 (2)	Shameer	2-4
Sanju	3 (6)	Krishnanunni	2-9
GRIDZ WON BY 29 RUNS			
Man of the match: Krishnanunni			SCL 2K19

ABOUT THE DATA SET

- Kaggle data set containing 6025 matches.
- With over 11 straight forward features. They are:
- Venue, round, home, innings 1 and innings 2 as **categorical**
- Innings1_runs, innings 1_wickets, innings_1_overs_batted, as **Continuous.**

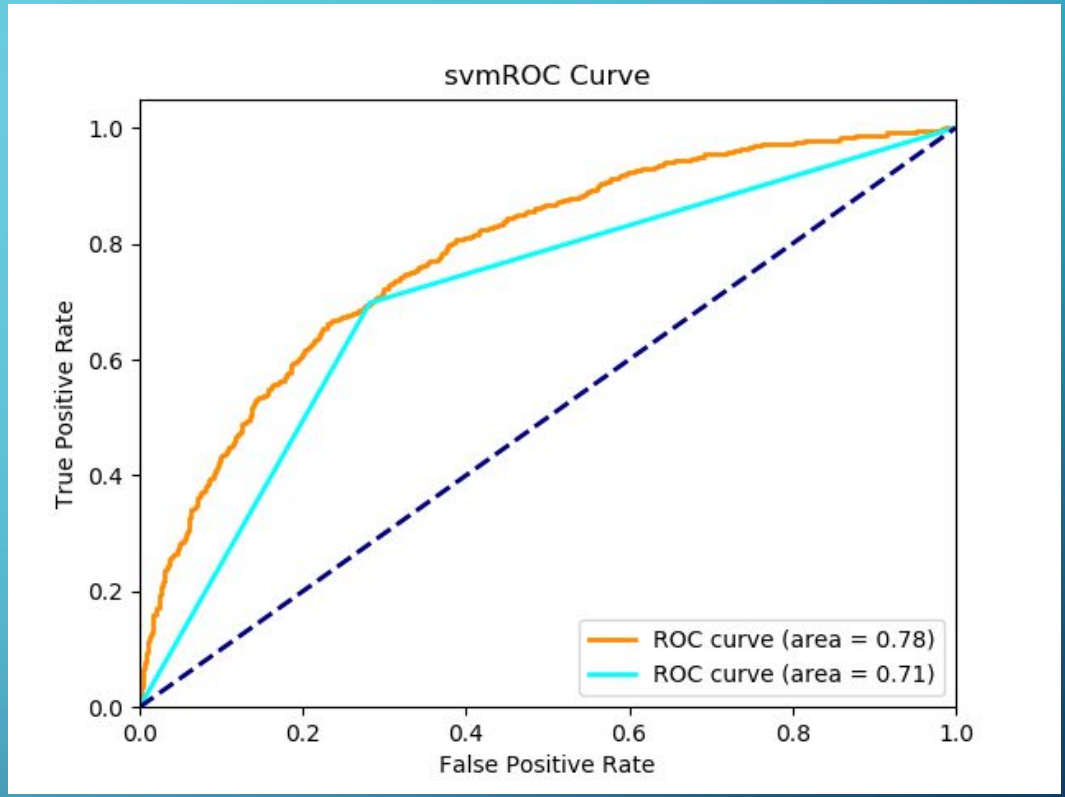
venue
round
home
away
winner
win_by_runs
win_by_wickets
balls_remaining
innings1
innings1_runs
innings1_wickets
innings1_wickets
innings1_overs_batted
innings1_overs

Support Vector Machine (SVM)

- Hyperparameters:
  - Kernel Function: ‘rbf’ (see below)
    - Radial Basis Function Kernel
  - gamma: ‘scale’
    - Scales coefficient used in rbf based on input dimensions
- Model Performance on Test Set:
  - Cohen’s Kappa: 0.415
  - Accuracy: 0.708

$$k(x_i, x_j) = exp\left(\frac{-1}{2}\text{distance}\left(\frac{x_i}{length\_scale}, \frac{x_j}{length\_scale}\right)^2\right)$$

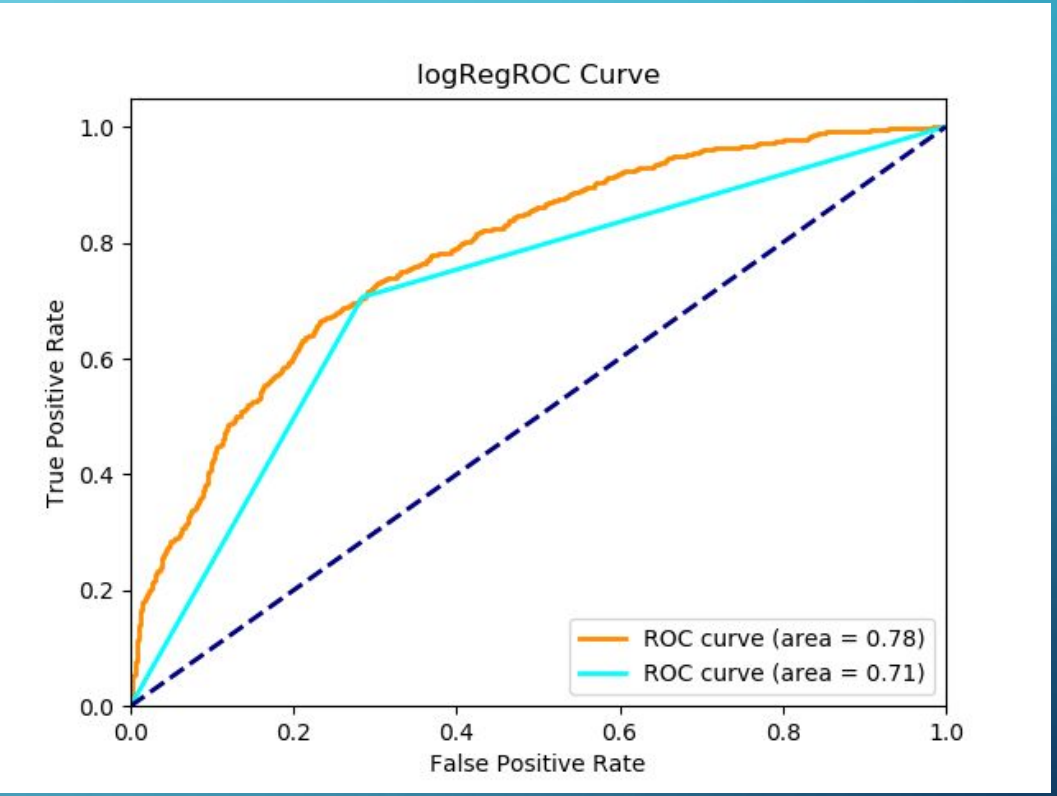
```
svcModel = GridSearchCV(
    estimator=SVC(
        gamma='scale',
        random_state=42
    ),
    param_grid={
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
    },
    n_jobs=-1,
    cv=5,
    scoring=make_scorer(cohen_kappa_score)
)
```



SVM Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	679	267
	False	262	605

Logistic Regression

- L1 Regularization (Lasso)
  - Built-in feature selection
- Model Performance on Test Set:
  - Cohen's Kappa: 0.421
  - Accuracy: 0.711
- *Highest Cohen Kappa/Accuracy Model*



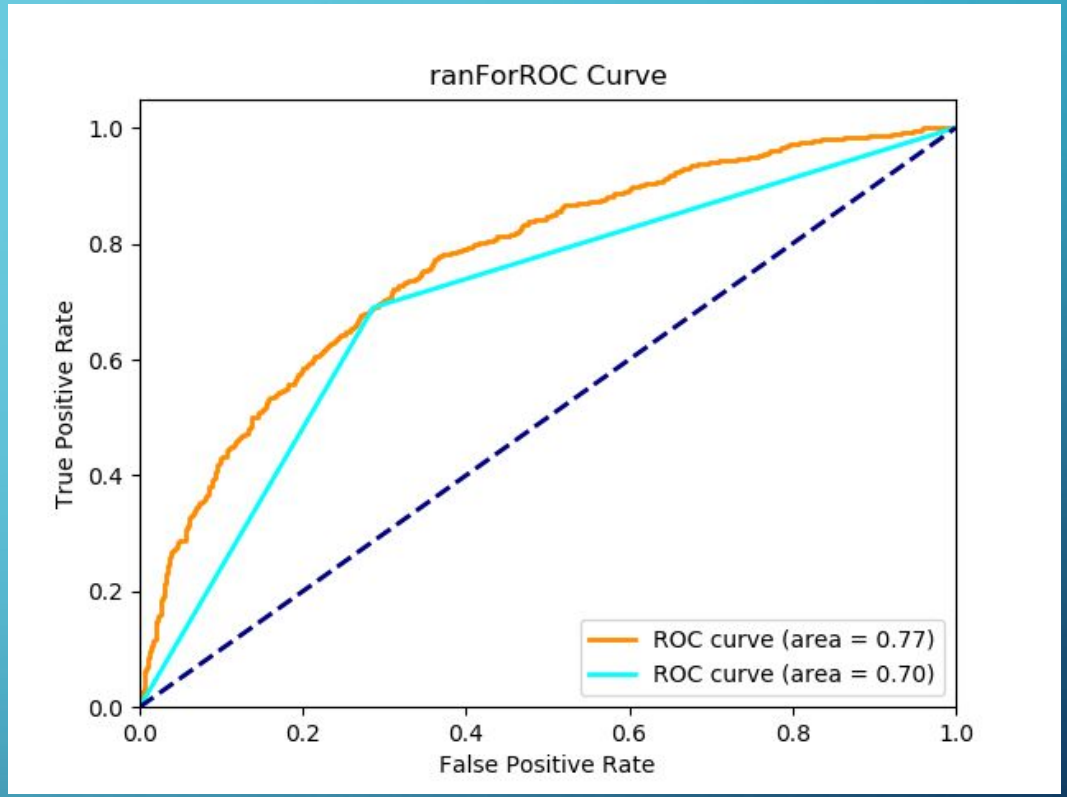

```
logRegModel = LogisticRegressionCV(
    max_iter=200,  # doubled default max_iter value to help with convergence
    tol=0.01,  # increase tolerance above default by factor of 10 to help
               #   with convergence.
    cv=5,  # Increased number of folds to 5 (default is 3)
    penalty='l1',  # Using l1 regularization as feature selection is implied
    solver='saga',  # Using saga solver as it supports l1 reg., and warm start
                    #   according to docs, warm start helps speed up CV loop
    random_state=42,
    scoring=make_scorer(cohen_kappa_score)
)
```

LogReg Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	677	271
	False	255	612

Random Forest

- Hyperparameters:
  - number of trees: 200
  - max depth of trees: 50
- Model Performance on Test Set:
  - Cohen's Kappa: 0.403
  - Accuracy: 0.702

```
ranForModel = GridSearchCV(
    estimator=RandomForestClassifier(
        random_state=42,
        n_jobs=-1
    ),
    param_grid={
        'n_estimators': [10, 100, 200, 500],
        'max_depth': [25, 50, 100],
    },
    n_jobs=1,
    cv=5,
    scoring=make_scorer(cohen_kappa_score)
)
```

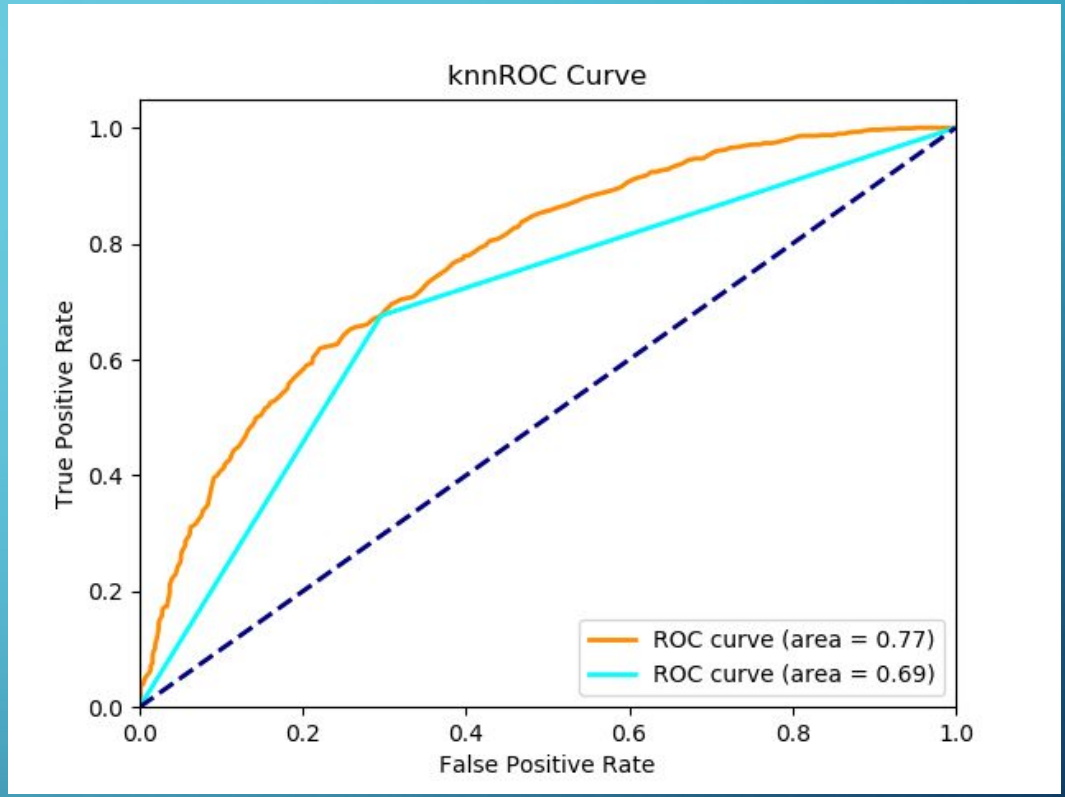


RF Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	675	271
	False	269	598

KNearestNeighbors

- Hyperparameters:
  - number of neighbors: 317
- Model Performance on Test Set:
  - Cohen's Kappa: 0.380
  - Accuracy: 0.691

```
knnModel = RandomizedSearchCV(
    estimator=KNeighborsClassifier(
        n_jobs=1
    ),
    param_distributions={
        'n_neighbors': sp_randint(100, 500)
    },
    n_iter=200,
    n_jobs=-1,
    cv=5,
    scoring=make_scorer(cohen_kappa_score),
    random_state=42
)
```



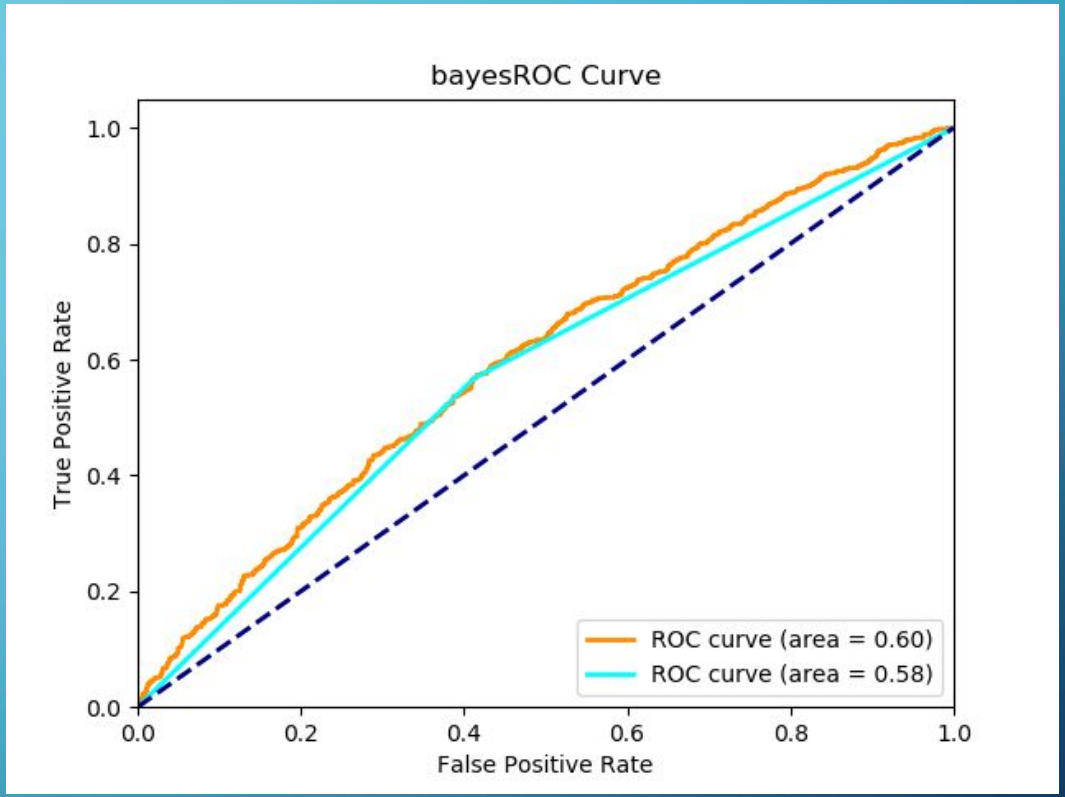
KNN Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	666	280
	False	281	586

Bernoulli Naive Bayes

- Only uses Categorical Variables!
- Model Performance on Test Set:
  - Cohen's Kappa: 0.155
  - Accuracy: 0.578

*NullAccuracyRate: 0.523

```
nbmodel = BernoulliNB()
```

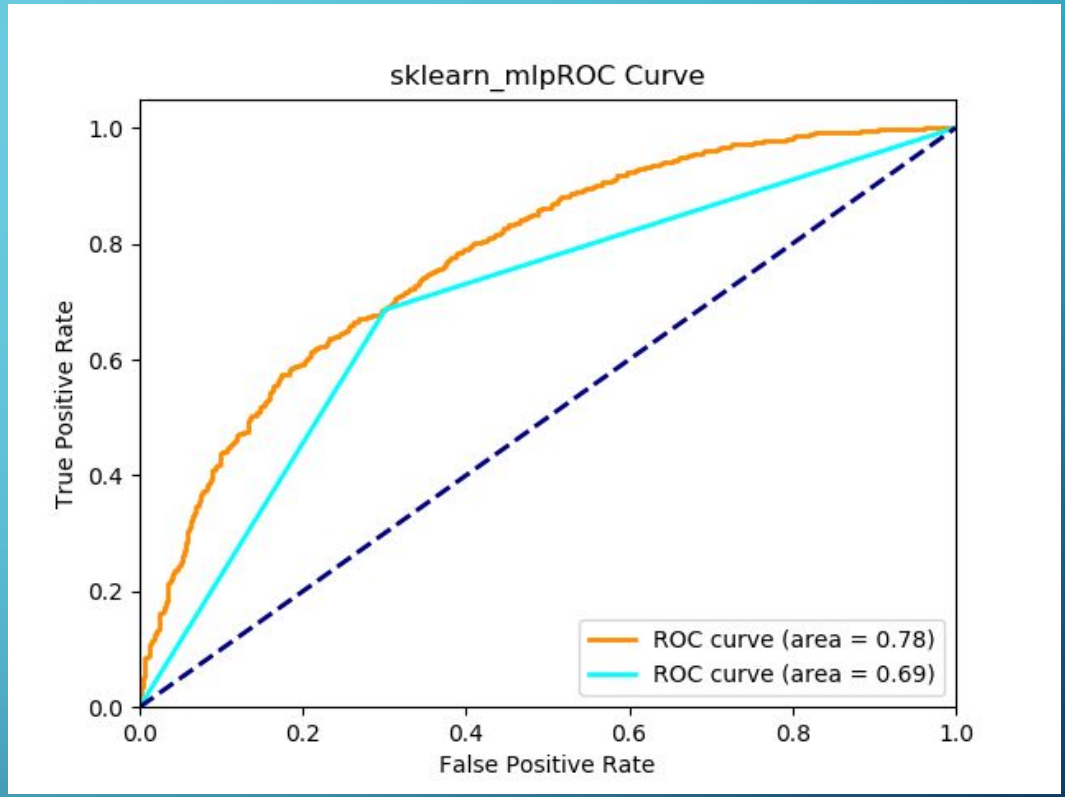


Bayes Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	556	390
	False	375	492

Sklearn based MLP Network

- Big takeaway - Slow to train!
- Hyperparameters:
  - activation function: 'logistic'
  - solver: Stochastic Gradient Descent
  - learning rate: Adaptive
  - number of hidden layers: 1
  - number of neurons per hidden layer: 200
  - number of iterations: 500
- Convergence issues in training/setup
- Model Performance on Test Set:
  - Cohen's Kappa: 0.385
  - Accuracy: 0.693

```
mlpModel = MLPClassifier(
        activation='logistic',
        solver='sgd',
        learning_rate='adaptive',
        hidden_layer_sizes=(200,),
        max_iter=500)
```



MLP Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	661	285
	False	272	595

Keras based MLP Network Design

- Setting up Keras based experiments was a little more involved
  - GPU Acceleration required installing
    - CUDA, and CUDA NN Software
    - Tensorflow Software
  - Using sklearn RandomizedSearch requires wrapping the keras model (see right)
  - backend.clear_session() took a few hours to figure out
- Note several additional hyperparameters we didn't manager to setup for search:
  - activation function of hidden layers
  - loss function
  - optimizer algorithms
  - metrics functions
  - batch_size, (though errors occurred when i tried)

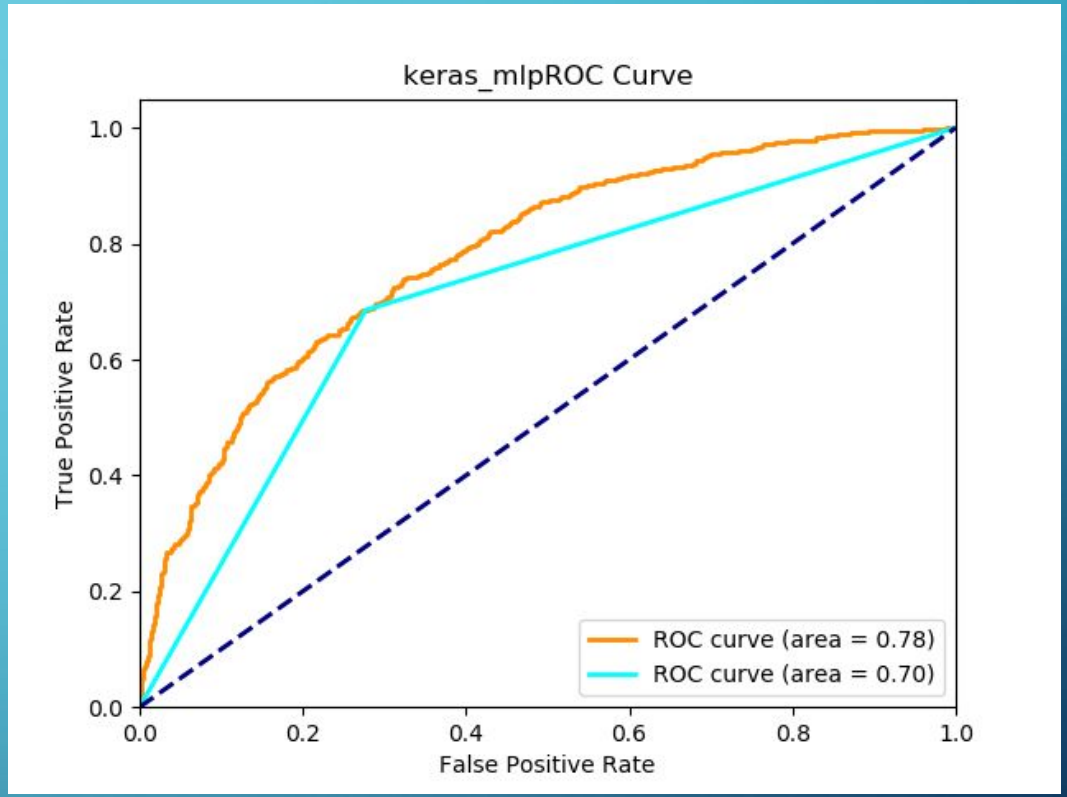
```
# inspired by https://keras.io/getting-started/sequential-model-guide/
#   and https://keras.io/scikit-learn-api/
def construct_model(dropout_rate=0.0,
                    extra_layers=1,
                    num_neurons=455):
    # num inputs: 1867
    #   seems to fix a memory issue.... no idea
    backend.clear_session()

    model = Sequential()
    model.add(Dense(num_neurons,
                    activation='relu',
                    input_dim=1867)),
    for i in range(extra_layers):
        model.add(Dropout(dropout_rate))
        model.add(Dense(num_neurons,
                        activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(2,
                    activation='softmax'))

    model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
estimator=KerasClassifier(
    build_fn=construct_model
),
# comments capture best setti
```

Keras based MLP Network

- Big takeaway - GPU Acceleration!
- Hyperparameters:
  - epochs: 1
  - dropout_rate: 0.32
  - extra_layers: 0
  - num_neurons: 476
- Model Performance on Test Set:
  - Cohen's Kappa: 0.409
  - Accuracy: 0.706

```
mlpModel = RandomizedSearchCV(
    estimator=KerasClassifier(
        build_fn=construct_model
    ),
    # comments capture best setting so far
    param_distributions={
        'epochs': sp_randint(1, 10),  # 3
        'dropout_rate': sp_uniform(0, 1),  # 0.8532567453865779
        'extra_layers': sp_randint(0, 15),  # 0
        'num_neurons': sp_randint(100, 960),  # 435
    },
    n_jobs=1,
    n_iter=10,
    cv=5,
    scoring=make_scorer(cohen_kappa_score),
    random_state=42
)
```



MLP Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	684	262
	False	273	594

Model Performance Summary Table

Model	Accuracy	Cohen's Kappa
Logistic Regression	0.711	0.421
Keras based MLP	0.706	0.409
Support Vector Machine	0.708	0.415
Random Forest	0.702	0.403
Sklearn based MLP	0.693	0.385
KNearestNeighbors	0.691	0.380
Bernoulli Naive Bayes	0.578	0.155

Stacked Models

- Transform existing dataset using output from the models we just trained
- Train various models to combine these new features for final predictions

Model	Accuracy	Cohen's Kappa
Stacked Bernoulli Naive Bayes	0.707	0.414
Stacked KNearestNeighbors	0.707	0.413
Stacked Random Forest	0.707	0.412
Stacked Logistic Regression	0.707	0.412
Stacked Support Vector Machine	0.707	0.412

CONCLUSION

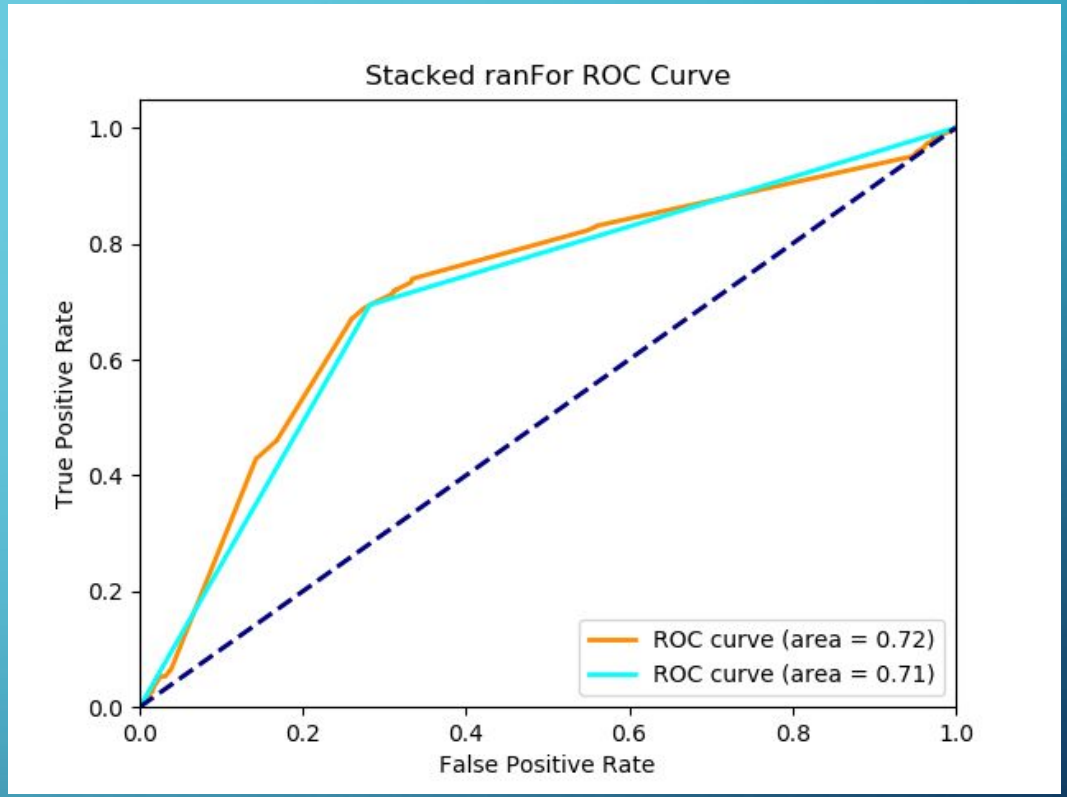
- Model performance on the test set capped out at just over 70 percent accuracy.
  - 70%-ish cap seems consistent between multiple models using a variety of techniques and assumptions
  - Seems to imply that the target is simply not predictable much beyond a 70 percent accuracy, given the predictors we've been able to collect.
  - Adding more features to the dataset, especially regarding team performance statistics, would likely help improve the accuracy of the model, and would likely be the next step in furthuring this research.
- Best Predictive Model overall was Logistic Regression (Lasso/L1 Regularization):
  - Accuracy: 0.711, Cohen's Kappa: 0.421
- Best Stacked Model was Bernoulli Naive Bayes:
  - Accuracy: 0.707, Cohen's Kappa: 0.414

Backup Slides

Stacked Random Forest

- Model Performance on Test Set:
  - Cohen's Kappa: 0.412
  - Accuracy: 0.707

```
def randomForest():
    return GridSearchCV(
        estimator=RandomForestClassifier(
            n_jobs=-1,
            random_state=42
        ),
        param_grid={
            'n_estimators': [5, 10, 20, 40, 80],
            'max_depth': [5, 25, 100, 200, None]
        },
        scoring=myscorer,
        cv=5
    )
```

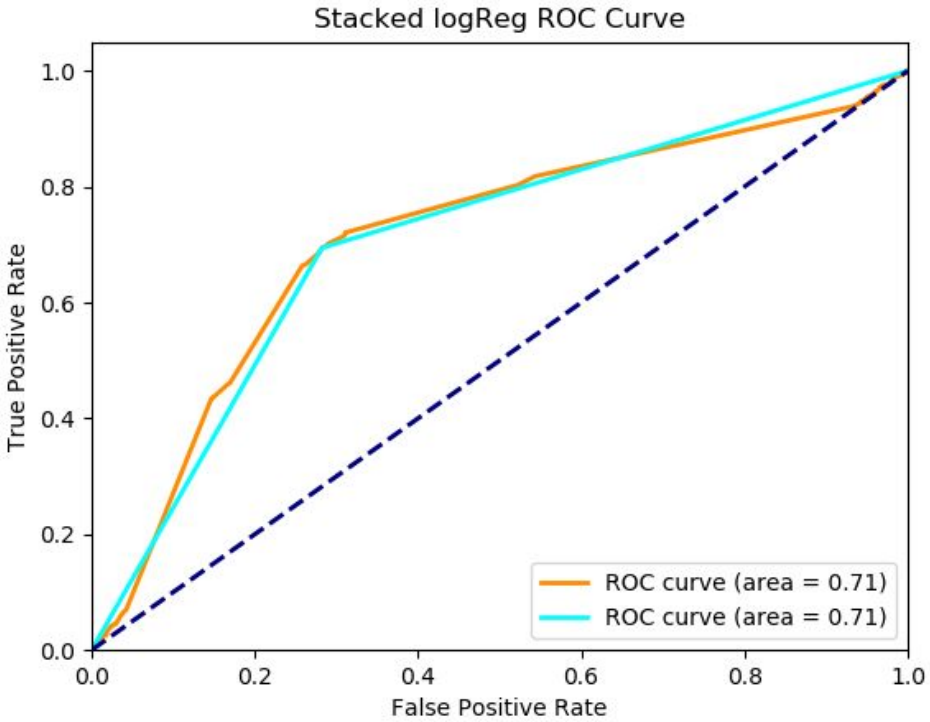


S RF Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	679	267
	False	265	602

Stacked Logistic Regression

- Model Performance on Test Set:
  - Cohen's Kappa: 0.412
  - Accuracy: 0.707

```
def logReg():
    return LogisticRegressionCV(
        max_iter=200,  # doubled default max_iter value to help with convergence
        tol=0.01,  # increase tolerance above default by factor of 10 to help
                   #   with convergence.
        cv=5,  # Increased number of folds to 5 (default is 3)
        penalty='l1',  # Using l1 regularization as feature selection is implied
        solver='saga',  # Using saga solver as it supports l1 reg., and warm start
                        #   according to docs, warm start helps speed up CV loop
        random_state=42,
        scoring=myscorer,
        n_jobs=-1
    )
```

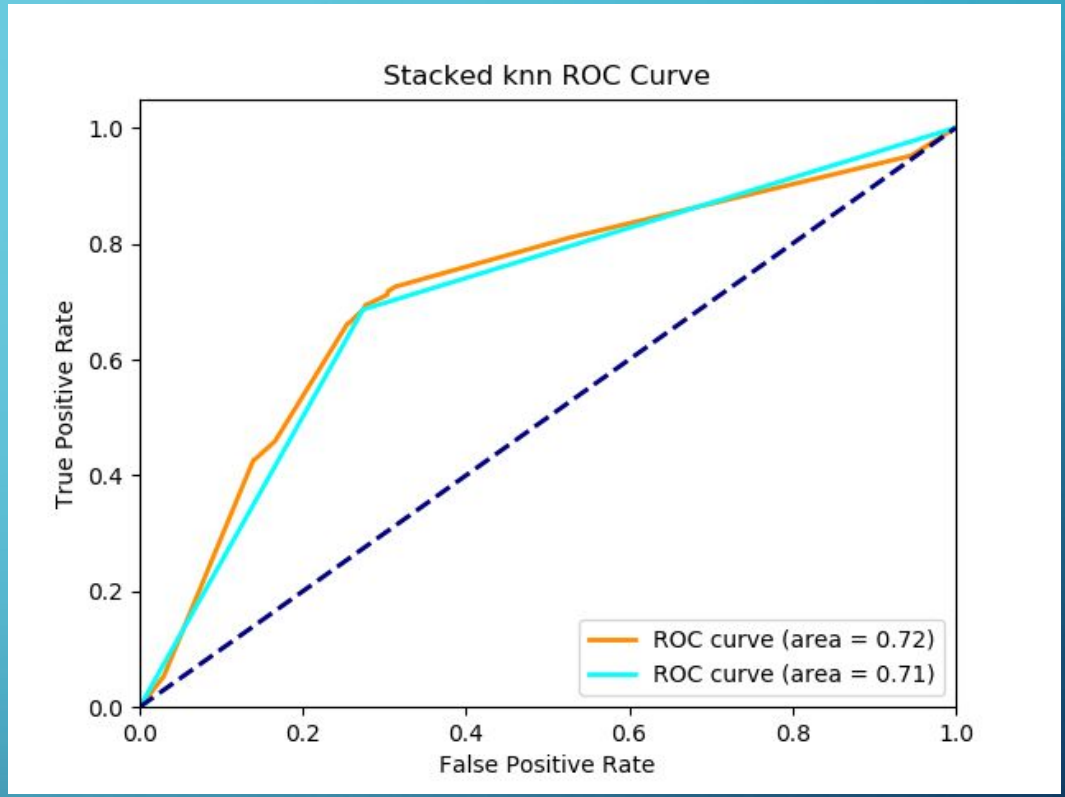


Stacked LogReg Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	679	267
	False	265	602

Stacked KNearestNeighbors

- Hyperparamters
  - n_neighbors: 21
- Model Performance on Test Set:
  - Cohen's Kappa: 0.413
  - Accuracy: 0.707

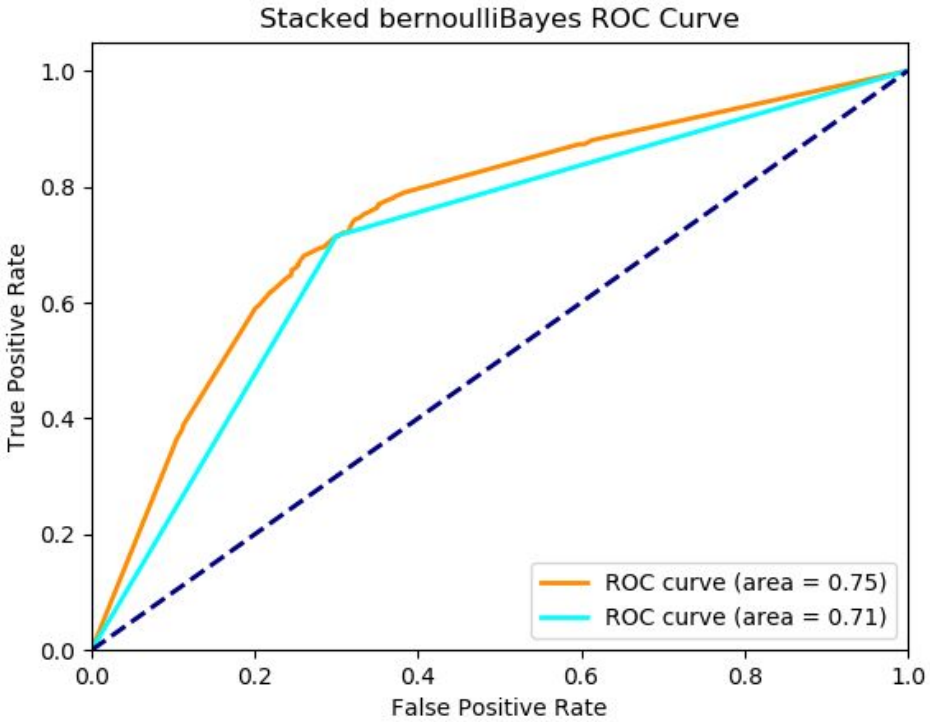
```
def knn():
    return RandomizedSearchCV(
        estimator=KNeighborsClassifier(
            n_jobs=-1
        ),
        param_distributions={
            'n_neighbors': sp_randint(1, 800)
        },
        n_iter=50,
        n_jobs=1,
        cv=5,
        scoring=myscorer,
        random_state=42
    )
```



Stacked KNN Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	687	259
	False	272	595

Stacked Bernoulli Naive Bayes

- Model Performance on Test Set:
  - Cohen's Kappa: 0.414
  - Accuracy: 0.707



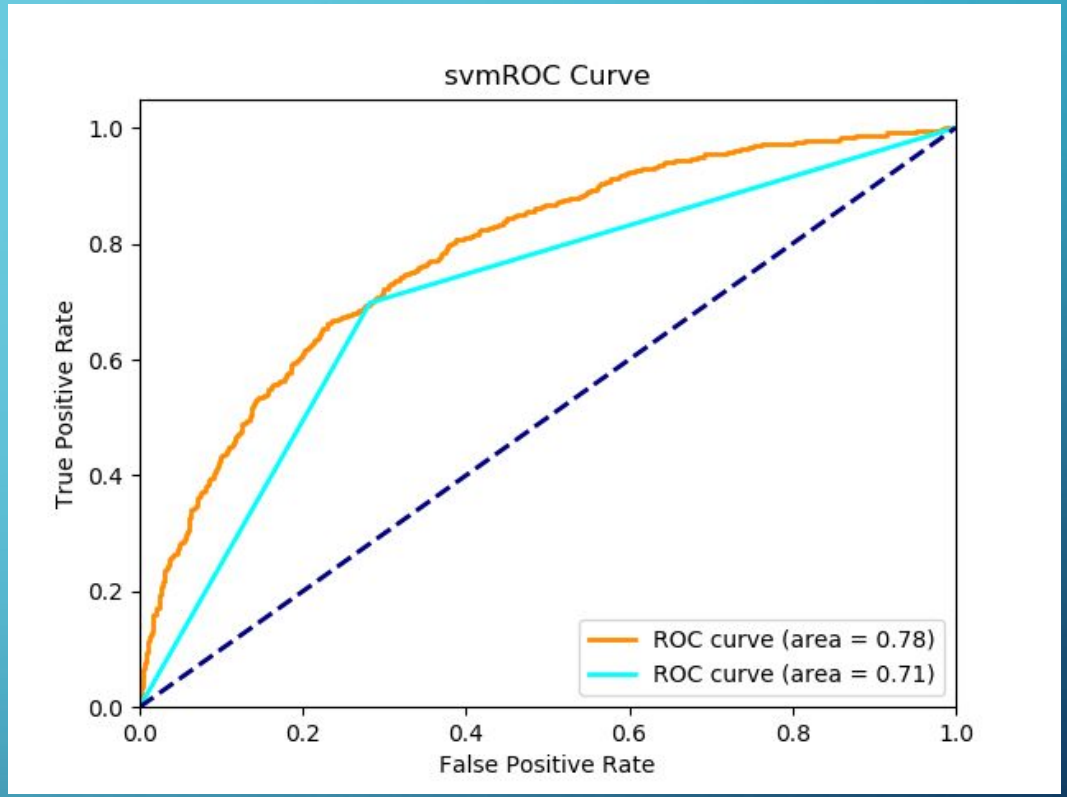

```
def bayes():
    return BernoulliNB()
```

Stacked KNN Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	662	284
	False	247	620

Stacked Support Vector Machine

- Model Performance on Test Set:
  - Cohen's Kappa: 0.412
  - Accuracy: 0.707

```
def svc():
    return GridSearchCV(
        estimator=SVC(
            gamma='scale',
            random_state=42
        ),
        param_grid={
            'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
        },
        n_jobs=-1,
        cv=5,
        scoring=myscorer
    )
```



Stacked KNN Confusion Matrix (vs. test data)		Actual Class	
		True	False
Predicted Class	True	679	267
	False	265	602