Predictions Using Machine Learning

Keertik Bacon

Centennial High School

4300 Centennial Lane

Ellicott City, MD 21042

Table of Contents

Introduction

Almost every week in the September to February NFL season, analysts from NFL.com, CBS, NBC, FOX, Yahoo Sports, local newspapers, and various other organizations make their predictions on the winners of the weekend's football games, with around 50 – 60 different takes on the game's outcome. These analysts base their predictions on previous performances, looking at the players, the coaching, the opposition, and other circumstances, to decide the winner, and the margin of victory. Since the performance of a team in future games is difficult to predict, and is contingent on many factors, predicting results has long been the domain of human analysts, as such relationships are too complex to condense into a logic expression format for computers to understand. However, with the advent of deep learning and neural networks, computers can be trained to find the pattern in a set of data without human intervention and use the pattern to make predictions. The objective of this research was to analyze the viability of machine learning algorithms in predicting complex occurrences, by creating an algorithm to predict the results of NFL games. It was hypothesized that the machine learning algorithm would be able to predict games with a success rate of at least 75%, higher than that of most analysts, whose rates are usually between 60% and 70%.

Materials

As this research project focuses on computer software and creating computer algorithms, the only equipment needed was a computer and several pieces of software.

The Python programming language was used to set the parameters for the creation and training of the machine learning algorithm, and make predictions using the input data and algorithm. Enthought Canopy was used as the IDE in which all the code, including that for training the algorithm, taking input data and adapting it for its own use, and making predictions with the data and algorithm, was written.

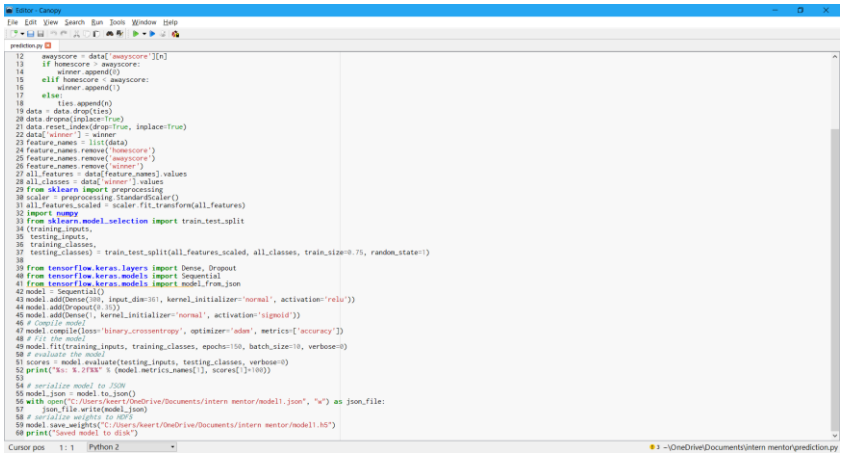


Figure 1. Screenshot of the Python program used to build the neural network.

TensorFlow and Keras were used in tandem to create the deep learning model, with TensorFlow used to store the data for each neuron, and Keras used on top to organize the data for the neurons into layers and creating a “network.”

Methods

A 1431 NFL game dataset was used to train the models. This dataset contains 60 statistics each for both teams in a game, averaged over three different time periods – the past season, the current season, and the past three games. The dataset does not contain the identities of either team participating, allowing the algorithm to be more flexible and predict matchups based on how similar teams performed against similar opponents, rather than looking how that specific team performs.

A variety of machine learning techniques were trained and tested using the dataset, to see which one works best in this situation. These techniques were:

- Decision Tree
- Random Forest
- Logistic Regression
- Support Vector Machine
- K Nearest Neighbors
- Naïve Bayes
- Neural Network

Each of these methods were used to create a model, with varying accuracies. All the models were optimized for maximum performance, except for the neural network, whose flexibility allows for much more optimization than the other techniques.

Results

Decision Tree

This technique analyzes the data and creates a flowchart that it consults to make predictions. Decision trees work better with simpler relationships with fewer variables and less overall noise in the data. Unfortunately, that means that this technique is not very suited for predicting NFL games, which are more complex.

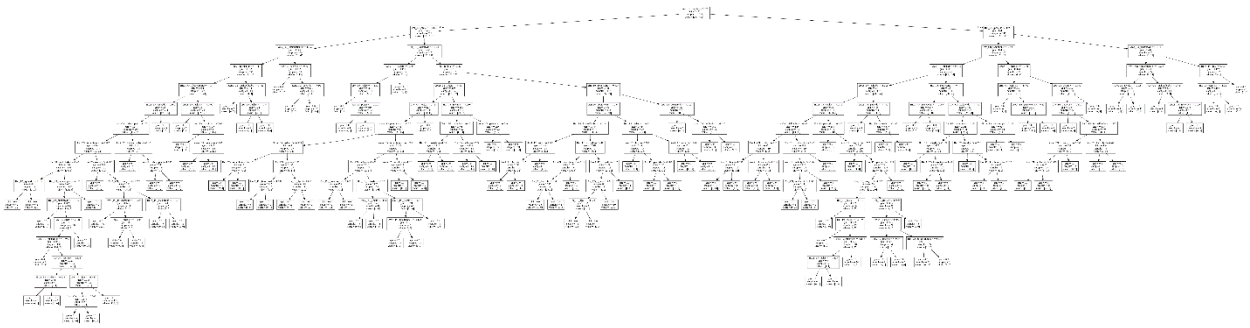


Figure 2. Decision tree created by the program to use for predicting results.

The decision tree model had the lowest accuracy of all the machine learning models, at 56.8%.

Random Forest

This technique builds off decision trees, building a “forest” of several decision tree models and having them vote on the prediction. This method is better, as it aggregates a prediction rather than relying on one model, but it is only marginally better than decision trees in this application, as it still suffers from the problem that it is not suited for a large and complex dataset. The Random Forest model had an accuracy of 59.9%.

Logistic Regression

This technique plots the data points and finds a logistic function that fits the data and can be used to predict additional points. This method is considerably more basic than other machine learning techniques, as it is similar to linear and quadratic regression, which are used for tightly correlated and simple relationships. Logistic functions have an additional advantage of being better at choosing a binary answer, due to their s-shaped curve.

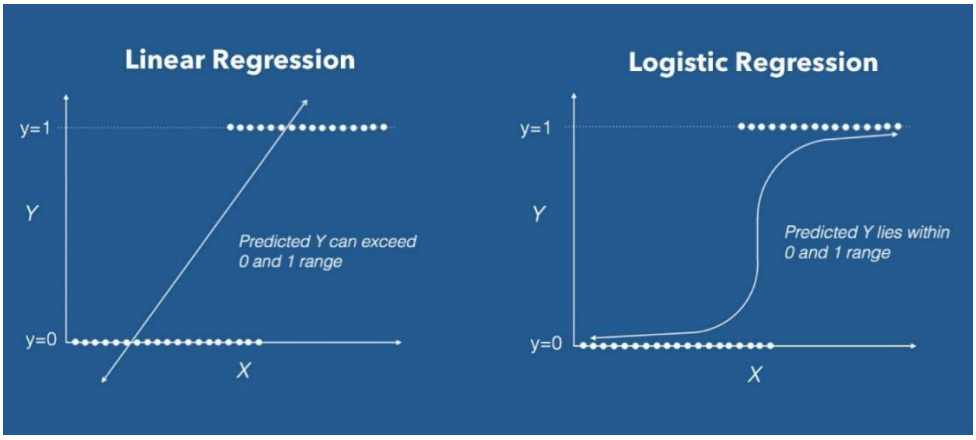


Figure 3. Comparison between linear and logistic regression; Retrieved on 9 Feb. 2019 from https://medium.com/greyatom/logistic-regression-89e496433063

However, the model could not handle the complex data, and it had an accuracy of only 59.8%.

Support Vector Machine (SVM)

This method plots all the training points and attempts to group all the points with the same classification (ex. match won/lost) together by drawing dividing lines between points of different classifications. This method allows for more complex relationships, as instead of focusing on modeling the actual data, it focuses on dividing up the graph between the different classifications.

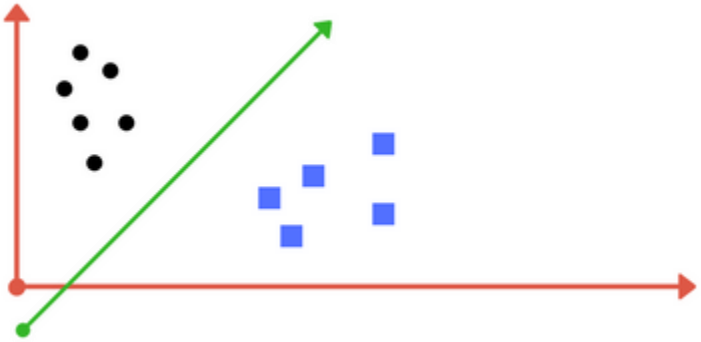


Figure 4. Diagram illustrating that support vector machines work by dividing up the graph between the different classes of data. Retrieved on 9 Feb. 2019 from https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

The SVM model performed considerably better than any of the previous models, with an accuracy of 63.4%.

K Nearest Neighbors

This technique plots all the training data points, but unlike SVM, does not draw lines to split up the graph based on classification. Instead, the algorithm plots the input point, finds the closest points to it on the graph, and uses their classification to determine that of the input point. Essentially, the algorithm finds the most similar points to the input point and passes on their classification to the input point. The number of closest neighbors that the algorithm looks at can be adjusted, as some numbers yield better results than others.

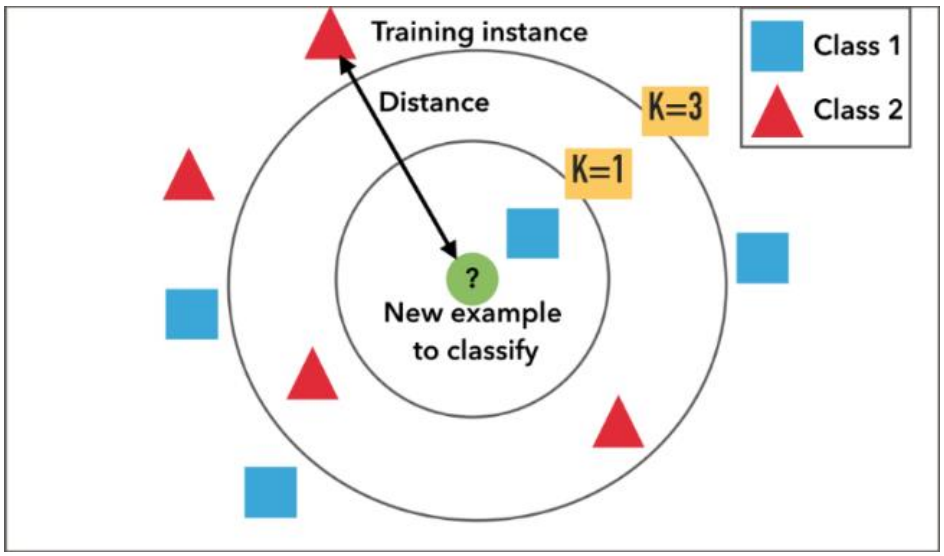


Figure 5. Diagram illustrating that K Nearest Neighbor algorithms work by looking at the classifications of points around the input point. Retrieved on 9 Feb. 2019 from https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

17 neighbors proved to be the most successful, giving the model a success rate of 63.5%, 0.1 percentage points better than the SVM model.

Naïve Bayes

This method uses probability to determine the classification of a data point, by analyzing the chance of a certain event occurring, given that another event has occurred. If certain statistics are input, and the algorithm notices that in the training set that teams with similar statistics have lost more often than they have won, the model will predict that the input team will lose. The technique is "naïve," because it assumes that all the independent variables are equally important, and independent of each other. Aside from this small caveat, the method is useful for modelling complex relationships.

The Naïve Bayes model had a success rate of 63.7%, which is quite remarkable compared to the other techniques. However, this places it along the lower end of 60 - 70% accuracy of analysts, meaning that it is still not quite on equal footing with humans.

Neural Networks

Neural networks are different from previous techniques, in that they offer much more potential and flexibility for optimization. They consist of a multitude of neurons, which are arranged in layers, and connected to many other neurons in different layers. Training the neural network consists of inputting values and adjusting the sensitivity of the connections to have the network output the correct classifications. The number of layers and neurons in each layer can be altered, allowing the technique flexibility and potential to improve.

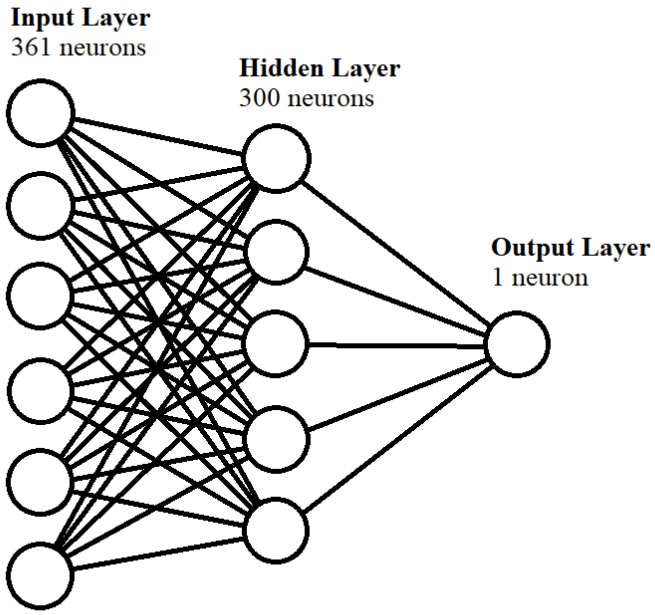
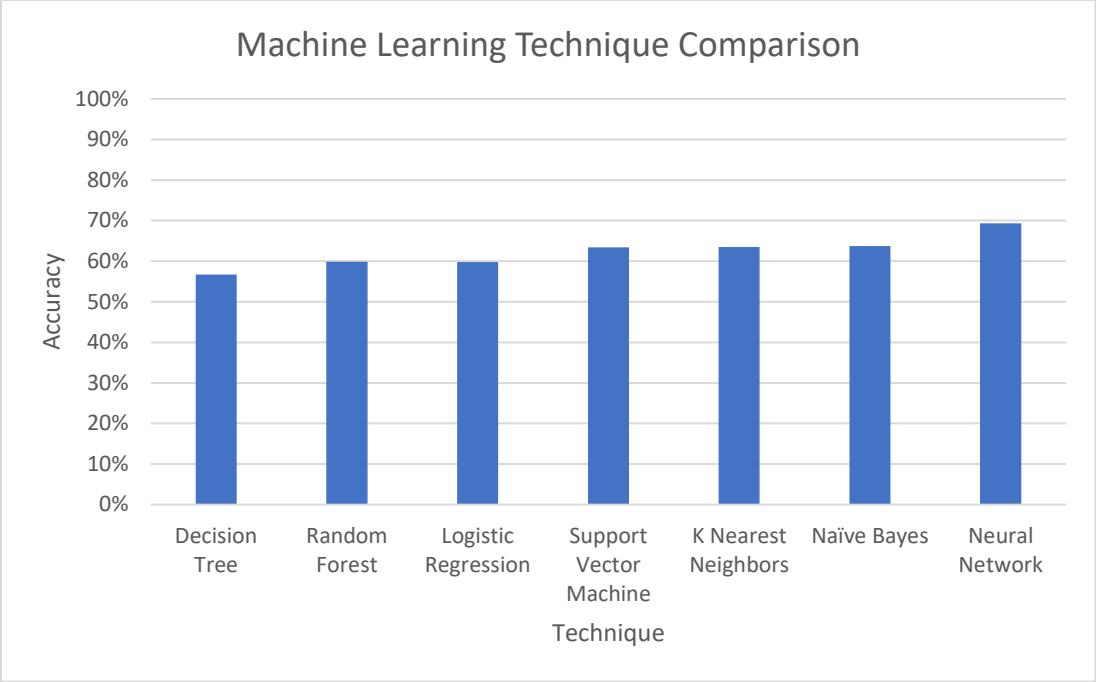


Figure 6. Diagram of the neural network used in this project.

The neural network used for this project consisted of 3 layers: an input layer consisting of 361 neurons (one for each statistic inputted), a hidden layer with 300 neurons, and an output layer with 1 neuron. The output neuron would output either a 0 or 1, predicting either a home team or away team victory.

The neural network was by far the most successful model, with an accuracy of 69.4%, paralleling the accuracies of some of the more successful NFL analysts in terms of match predictions.



Graph 1. A comparison of the success rate of all the different machine learning techniques used in this research project. The model with the lowest accuracy was the decision tree, with 56.8%, and the model with the highest accuracy was the neural network, with 69.4%.

Conclusion

The neural network was successful in supporting the viability of machine learning algorithms in predicting NFL matchups, proving its mettle by predicting with an accuracy on par with that of the top analysts. During the 2018 – 19 NFL season, the algorithm was used to predict a few game results, and it had an accuracy of 100% in such predictions, even predicting two upsets. However, the sample size is limited to three games, and so definitive conclusions cannot be made, making the 69.4% success rate still the most accurate metric. With some additional optimization, the model could reach the originally hypothesized accuracy of 75%.

Several aspects of both the dataset and the model can be improved to achieve this goal. For example, the year of each game could be added to the dataset, as in more recent years a higher offensive output is necessary to win. Another statistic that could be added is quarterback interceptions, as a team that is intercepted more usually loses.

The neural network used in this project is a base model. There are other derivatives of the neural network that are better at certain applications. For example, the Recurrent Neural Network can analyze a series of input data, making it well-suited for analyzing a series of games without needing to average the statistics, as was done when feeding them into the base neural network used in this project. In addition, the RNN would be more flexible with weighting more recent results over earlier ones, as was clumsily done with the three time periods used for the base network.

Unfortunately, both proposals would require significant overhaul and possibly replacement of the 1431 game training dataset, to add the required statistics and tailor it for an RNN. But if it succeeds in increasing the accuracy of the algorithm, it will not be in vain.

This machine learning technology is not just suited for predicting NFL games, and the code to create an algorithm is quite versatile. By switching out the training dataset and performing a few optimizations, the algorithm can easily be adapted to predict occurrences such as floods, an especially relevant topic in Ellicott City. The town has suffered two “100-year floods” in the past 3 years, and thus could benefit from an algorithm to predict floods, either through advising evacuation, or suggesting fixes for the infrastructure to mitigate flooding. By doing this, machine learning could quite literally save lives.

Appendix 1 – Works Cited

"A Machine Learning Analysis Of The NFL: Predicting New Playoff Contenders." *The Harvard Sports Analysis Collective*, 31 Dec. 2018, harvardsportsanalysis.org/2018/09/a-machine-learning-analysis-of-the-nfl-predicting-new-playoff-contenders/.

Blanchard, Ross. "Machine Learning for NFL Game Prediction: 2017 Season Retrospective." *Becoming Human: Artificial Intelligence Magazine*, Medium, 4 Feb. 2018, becominghuman.ai/machine-learning-for-nfl-game-prediction-2017-season-retrospective-cfda3ea66a3d.

Brid, Rajesh S. "Logistic Regression." *Medium*, 17 Oct. 2018, medium.com/greyatom/logistic-regression-89e496433063.

Bronshtein, Adi. "A Quick Introduction to K-Nearest Neighbors Algorithm." *Medium*, 11 Apr. 2017, medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7.

"Exploring Naïve Bayes Classifier: Maybe Not so Naïve after All?!" *Provalis Research*, 11 Aug. 2017, provalisresearch.com/blog/exploring-naive-bayes-classifier-maybe-not-naive/.

Patel, Savan. "Chapter 2 : SVM (Support Vector Machine) - Theory." *Medium*, 3 May 2017, medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

"Predict NFL Scores." *Kaggle*, Kaggle, www.kaggle.com/c/predict-nfl-game-scores-lawfty-dsco17/data.

"Pro Football Statistics and History." *Pro-Football-Reference.com*, www.pro-football-reference.com/.

Appendix 2 – Python Code

Algorithm class – gathers training data and builds algorithm

```
import pandas as pd
from sklearn.model_selection import cross_val_score


data = pd.read_csv('C:/Users/keert/OneDrive/Documents/intern mentor/hacknight_train.csv')
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
# Replace the columns with the score with winner column
# 0 for home team winning, 1 for away team winning
winner = []
ties = []
for n in range(0, data.shape[0]):
homescore = data['homescore'][n]
awayscore = data['awayscore'][n]
if homescore > awayscore:
winner.append(0)
elif homescore < awayscore:
winner.append(1)
else:
ties.append(n)
data = data.drop(ties)
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
data['winner'] = winner
feature_names = list(data)
feature_names.remove('homescore')
feature_names.remove('awayscore')
feature_names.remove('winner')
all_features = data[feature_names].values
```

```
all_classes = data['winner'].values
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(all_features)
import numpy
from sklearn.model_selection import train_test_split
# Split data up into training and testing sets
# 75% is training data, 25% is testing data
(training_inputs,
testing_inputs,
training_classes,
testing_classes) = train_test_split(all_features_scaled, all_classes, train_size=0.75,
random_state=1)

from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import model_from_json
# Make a neural network with 361 input neurons, 300 hidden layer neurons, and 1 output
model = Sequential()
model.add(Dense(300, input_dim=361, kernel_initializer='normal', activation='relu'))
model.add(Dropout(0.35))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(training_inputs, training_classes, epochs=150, batch_size=10, verbose=0)
# Evaluate the model
scores = model.evaluate(testing_inputs, testing_classes, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# Save model to file
```

```
model_json = model.to_json()
with open("C:/Users/keert/OneDrive/Documents/intern mentor/model1.json", "w") as json_file:
json_file.write(model_json)
# Save neuron connection weights as well
model.save_weights("C:/Users/keert/OneDrive/Documents/intern mentor/model1.h5")
print("Saved model to disk")
```

Predictor class – loads saved model and input data, and makes a prediction

```
from tensorflow.keras.models import model_from_json

import pandas as pd

data = pd.read_csv('C:/Users/keert/OneDrive/Documents/intern mentor/falconsravens.csv')
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)

# load model from file
json_file = open('C:/Users/keert/OneDrive/Documents/intern mentor/model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into model
loaded_model.load_weights("C:/Users/keert/OneDrive/Documents/intern mentor/model.h5")
print("Loaded model from disk")
# evaluate loaded model on test data
loaded_model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
prediction = loaded_model.predict_classes(data)
print(prediction)
```

Appendix 3 – Code for Previous Algorithms

```
import pandas as pd
from sklearn.model_selection import cross_val_score

data = pd.read_csv('C:/Users/keert/OneDrive/Documents/intern mentor/hacknight_train.csv')
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
# Replace the columns with the score with winner column
# 0 for home team winning, 1 for away team winning
winner = []
ties = []
for n in range(0, data.shape[0]):
homescore = data['homescore'][n]
awayscore = data['awayscore'][n]
if homescore > awayscore:
winner.append(0)
elif homescore < awayscore:
winner.append(1)
else:
ties.append(n)
data = data.drop(ties)
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
data['winner'] = winner
feature_names = list(data)
feature_names.remove('homescore')
feature_names.remove('awayscore')
feature_names.remove('winner')
all_features = data[feature_names].values
all_classes = data['winner'].values
```

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(all_features)
import numpy
from sklearn.model_selection import train_test_split
# Split data up into training and testing sets
# 75% is training data, 25% is testing data
(training_inputs,
testing_inputs,
training_classes,
testing_classes) = train_test_split(all_features_scaled, all_classes, train_size=0.75,
random_state=1)

# [Code for specific technique goes here]

# Builds algorithm, outputs score
cv_scores = cross_val_score(clf, all_features_minmax, all_classes, cv=10)

cv_scores.mean()
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
clf= DecisionTreeClassifier(random_state=1)
all_features_minmax = all_features_scaled
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10, random_state=1)
all_features_minmax = all_features_scaled
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
```

SVM

```
from sklearn import svm
clf = svm.SVC(kernel='rbf', C=C)
all_features_minmax = all_features_scaled
```

K Nearest Neighbors

```
from sklearn import neighbors
clf = neighbors.KNeighborsClassifier(n_neighbors=17)
all_features_minmax = all_features_scaled
```

Naïve Bayes

```
from sklearn.naive_bayes import MultinomialNB
scaler = preprocessing.MinMaxScaler()
all_features_minmax = scaler.fit_transform(all_features)
clf = MultinomialNB()
```