INTRODUCTION TO

Neural Networks in Python

Sabriya Alam and Parker Crain
Purdue University
Dec. 19th 2019

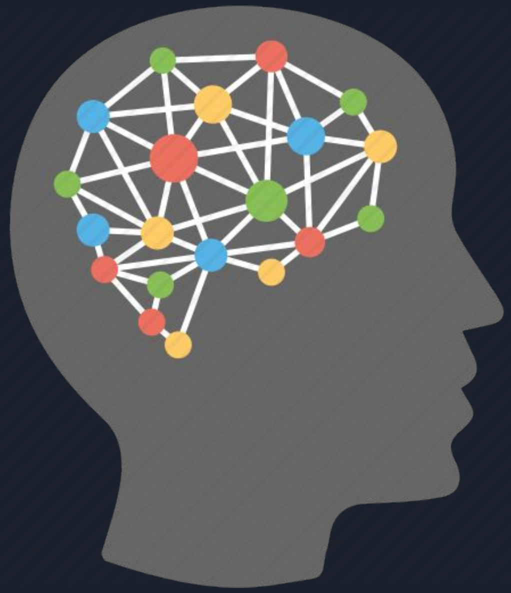
Outline

- Neural networks overview
- Data preprocessing techniques
- Base neural network model
- Network tuning
- Model metrics and evaluation

Neural Network Overview

https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

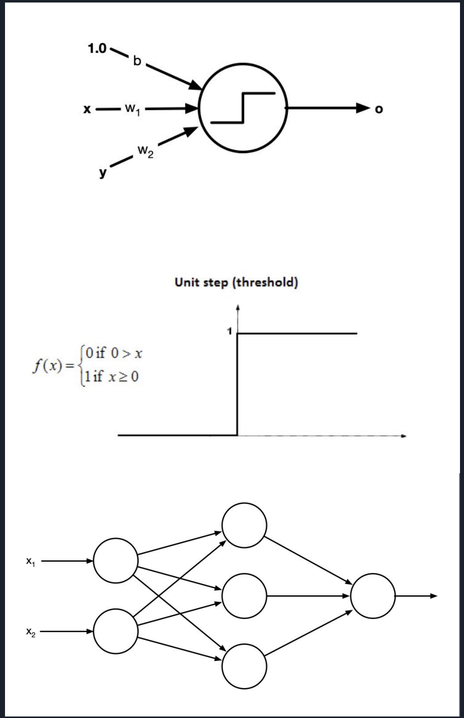
- What is an artificial neural network?
  - Machine learning computational model
  - Inspired by how biological neural networks in our brains process information and draw conclusions.
- Neural networks are used for
  - Speech recognition
  - Computer vision (image / video processing)
  - Text processing
  - Data analytics
- Many types of neural networks exist
  - Our focus will be on multi-layer perceptron



Neural Network Overview

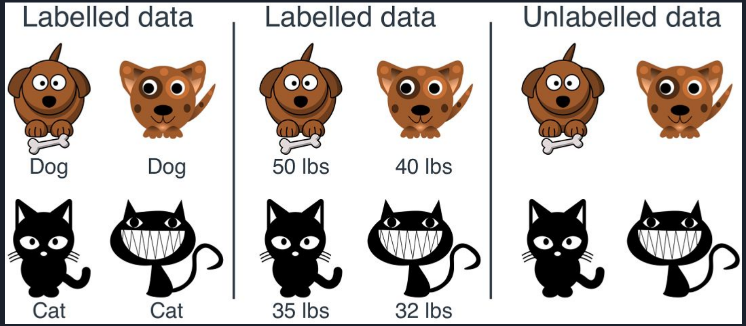
https://engineering.purdue.edu/~milind/ece20875/2019fall/notes/lecture-22.pdf

- Goal is to be able to learn non-linear classification boundaries
- Fundamental building block is a neuron (or perceptron)
  - Node with inputs and weights
  - Dot product is input to activation function
  - Output of neuron is result of activation function
- Single perceptron will converge if linear decision boundary exists. Multi-layer required for non-linear.
- Multi-layer = outputs of neurons become inputs to another layer of neurons
- Error in predicted values are back-propagated to adjust weights in the correct direction



Data Preprocessing Techniques

- Data comes in two types:
  - Labelled
    - Used for supervised machine learning
    - Known outcomes
  - Unlabelled
    - Used for unsupervised (deep) learning
    - Finds patterns without knowing how many classes there are or what class each data point belongs to
    - Examples: audio recordings, image data sets, etc.

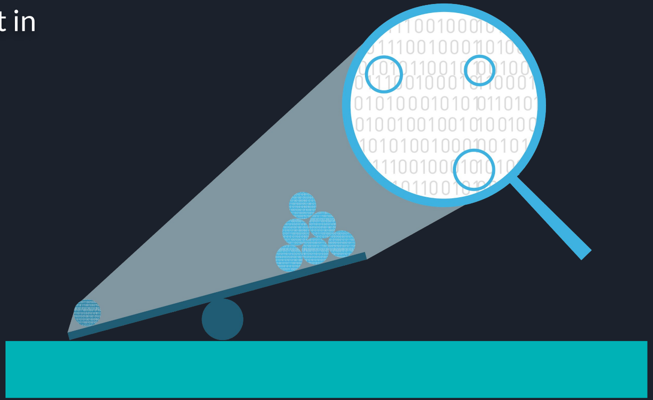


- Our data set is labelled

Data Preprocessing : Balancing Data

https://datascience.aero/predicting-improbable-part-1-imbalanced-data-problem/

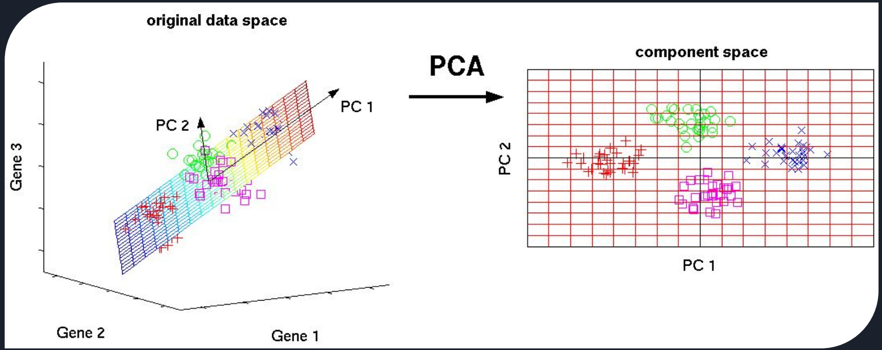
- Data needs to be divided into training and testing sets
- Eliminate bias (skewed results) by creating evenly balanced sets
  - If training set is 70% class A and 30% class B, the model may be biased (at least initially) toward predicting class A
  - If you know that this proportion is how the real world works, and the probability of event A occurring truly is 70%, then this is a relevant factor that should be kept in
  - Otherwise, create balanced sets with 50% class A and 50% class B



Data Preprocessing : Dimensionality Reduction

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

- Having a lot of input factors can slow down a model
- To make it faster, eliminate "duplicate" variables
  - Linearly correlated variables do not give any additional information, so keep only one
- Many statistical techniques for this
  - lasso regression
  - principal component analysis
- Using PCA brought our relevant features from 89 to 60



Code : PCA of balanced data sets

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
```

Import packages for normalizing data and performing PCA

```python
def preprocess(filename, filename2):
    dataset = pd.read_csv(filename, skiprows=2)
    training_array = np.array(dataset)
    X_train = training_array[:, 1:90]

    testset = pd.read_csv(filename2)
    test_array = np.array(testset)
    X_test = test_array[:, 1:90]

    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)
```

Import training data from *filename* as shown in previous presentation

Import testing data from *filename2*

PCA requires normalized data, so use scalar to make adjustments

```python
    pca = PCA(.95)
    pca.fit(X_train)
    n = pca.n_components_

    train_pca =   pca.transform(X_train)
    test_pca = pca.transform(X_test)

    return train_pca, test_pca, n
```

Create PCA model that will minimize number of parameters while still preserving 95% of the variance in the data

Apply PCA transformation to the two data sets

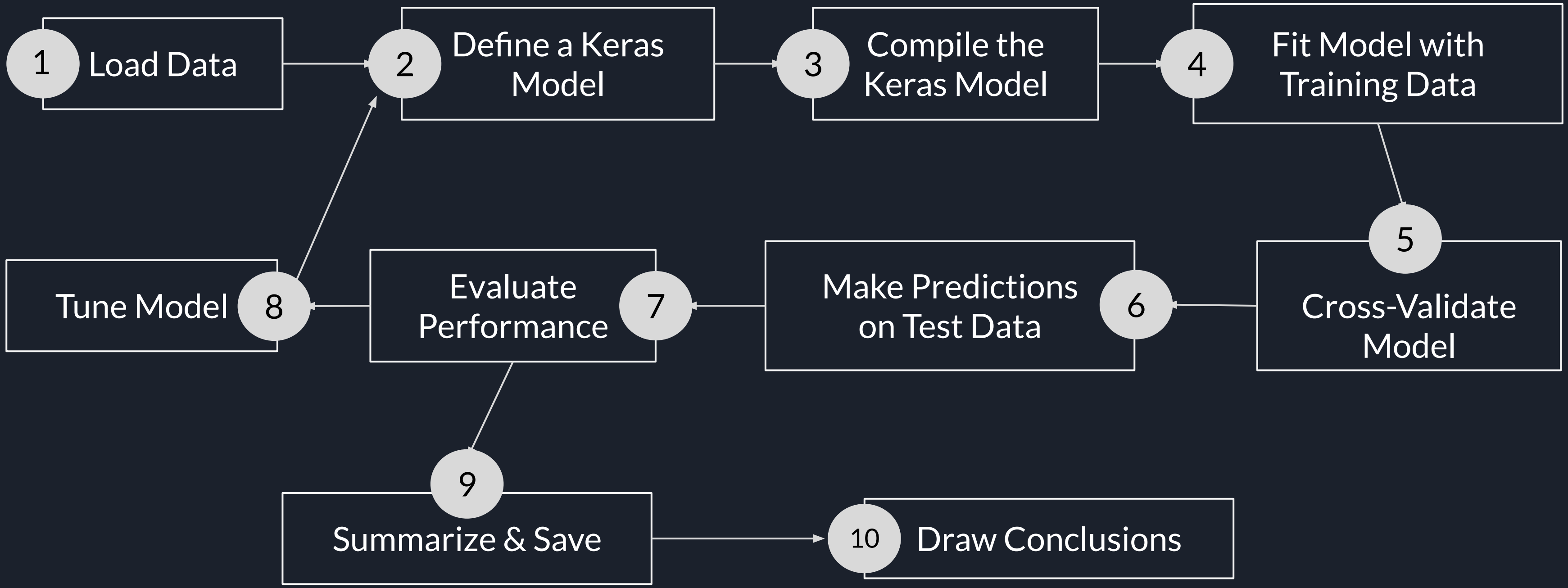
Neural Network Libraries in Python

https://towardsdatascience.com/introduction-to-deep-learning-with-keras-17c09e4f0eb2
https://medium.com/@Vatsal410/keras-without-nvidia-gpus-with-plaidml-and-amd-gpu-4ba6f60025ce

- Many options for neural network libraries in Python
- We chose to use Keras
    - Open-source
    - Can run on top of TensorFlow, R, PlaidML, Microsoft Cognitive Toolkit, Theano, etc.
    - User friendly
- Default backend is TensorFlow
- Without an Nvidia GPU, it is better to use PlaidML for accelerated performance

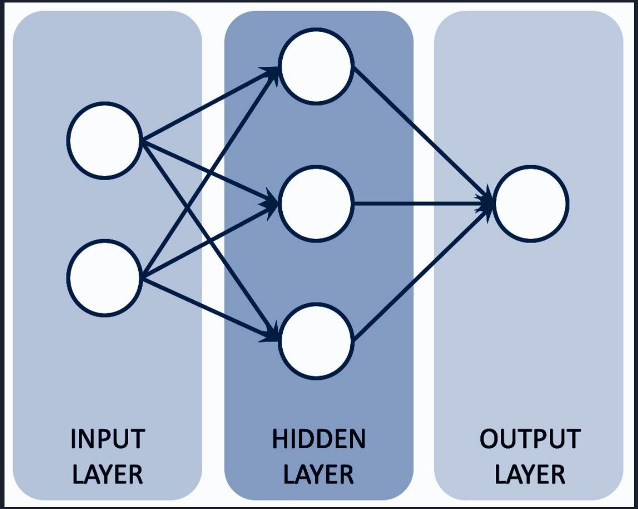
General Workflow

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/



Base Neural Network Model

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/



- There are no rules of thumb for choosing neural network model parameters.
  - Unique to each data set
  - Depends on data scope, structure, diversity and complexity
- Start with some randomized values, and then tune the model based on its performance.
- Base neural network is usually an input layer, one hidden layer, and one output layer.

Base Neural Network Model

Set up PlaidML and Keras, and import necessary modeling tools

```
import plaidml.keras
plaidml.keras.install_backend()
from keras.models import Sequential
from keras.layers import Dense
```

Read in training and testing data, along with expected outputs

```
def main(filename, filename2):
    n = 89
    dataset = pd.read_csv(filename, skiprows=2)
    new_array = np.array(dataset)
    X = new_array[:,1:90]
    y = new_array[:, 90]
    dataset = pd.read_csv(filename2)
    new_array2 = np.array(dataset)
    X_test = new_array2[:, 1:90]
    y_test = new_array2[:, 90]
```

Terminology:

Sequential - model is linear stack of layers

Dense - densely connected NN layer, so every node of this layer will connect to every node of the next layer

3 Layers :
Input, Hidden, and Output

As the model is being built, the accuracy is being cross validated with subsections of the training data too

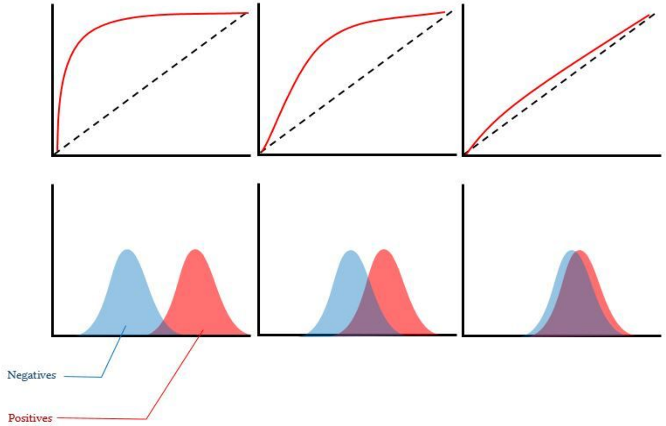
```
    model = Sequential()
    model.add(Dense(10, input_dim=n, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X, y, validation_split=0.33, epochs=5, batch_size=10)
    _, accuracy = model.evaluate(X, y)
    print('Accuracy: %.2f' % (accuracy * 100))
```

Model Performance Metrics & Evaluation

https://androidkt.com/get-the-roc-curve-and-auc-for-keras-model/

- Some metrics used for evaluating neural networks
  - ROC curve - plots false positive (x) and true positive (y) rates at various thresholds
    - Diagonal is equal to random guessing
    - Area under curve approaches 1 for perfect classifier
  - Confusion matrix
  - Precision, recall, and F1 scores
- Keras has built in functions to generate these

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)



$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{accuracy} = \frac{TP + TN}{TP + FN + TN + FP}$$

$$\text{specificity} = \frac{TN}{TN + FP}$$

Basic Model : Evaluation Metrics

(Function for plotting ROC)

```
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()
```

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score
from keras.models import load_model
```

Can view node weights

Model summary shows number of layers and nodes

```
print(model.get_weights())
print(type(model.get_weights()))
print("Model Summary:")
print(model.summary())
scores = model.evaluate(X, y, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1],
scores[1] * 100))
```

Save model, so you do not need to regenerate it every time

```
# save model and architecture to file
model.save("modelbasic.h5")
print("Saved model to disk")
```

```
print("loading model")
# load model
model = load_model('modelbasic.h5')
```

Input test data and get predictions from the model.

Print for comparison.

```
# make class predictions with the model
pred = model.predict_classes(X_test)
for i in range(0, len(pred)):
    print('%d (expected %d)' % (pred[i],
y_test[i]))
```

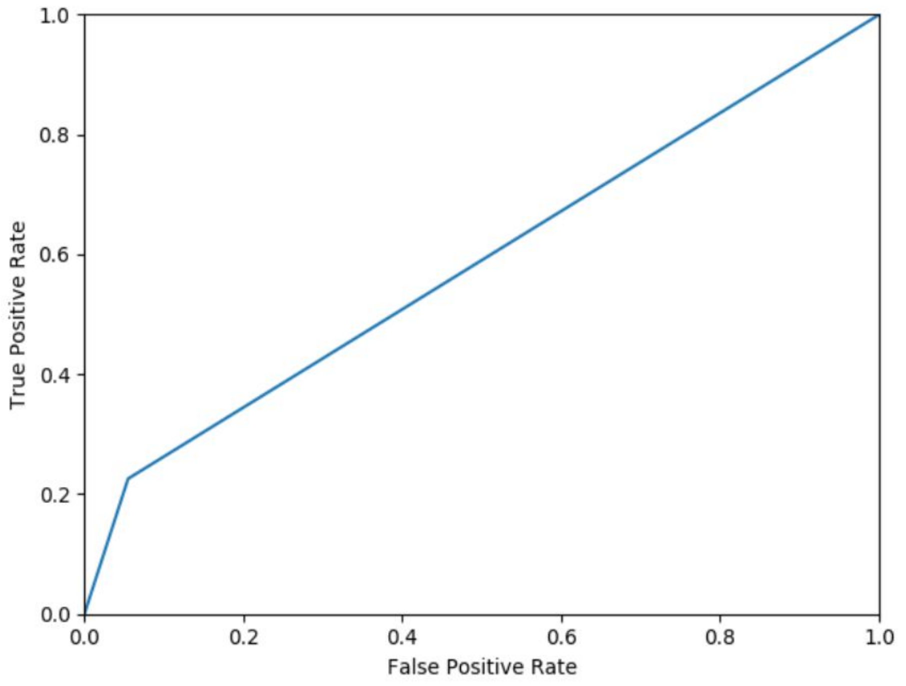
Calculate false positive and true positive rates for plotting ROC and getting AUC score

```
fpr, tpr, thresholds = roc_curve(y_test, pred)
plot_roc_curve(fpr, tpr)
print("auc score")
auc_score = roc_auc_score(y_test, pred)
print(auc_score)
```

Get precision, recall, f1-score, and confusion matrix.

```
print(classification_report(y_test, pred))
print("confusion matrix:")
print(confusion_matrix(y_test, pred))
```

Basic Model Evaluation Statistics



Accuracy: 60.02%

AUC Score = 0.5853871128871129

Classification Summary

	precision	recall	f1-score	support
0.0	0.55	0.94	0.69	1000
1.0	0.80	0.23	0.35	1001
accuracy			0.59	2001
macro avg	0.68	0.59	0.52	2001
weighted avg	0.68	0.59	0.52	2001

Confusion Matrix:

[[974 26]
 [835 166]]

Interpretation

Accuracy: 60.02%
AUC Score = 0.5853871128871129

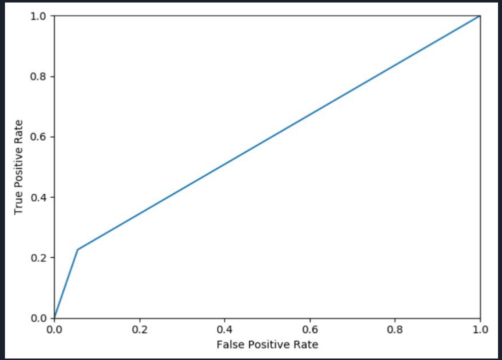
Classification Summary

	precision	recall	f1-score	support
0.0	0.55	0.94	0.69	1000
1.0	0.80	0.23	0.35	1001
accuracy			0.59	2001
macro avg	0.68	0.59	0.52	2001
weighted avg	0.68	0.59	0.52	2001

Confusion Matrix:
[[974 26]
 [835 166]]

- If the model says the patient will not be readmitted, probability that they will actually not be readmitted is .55
- If the model says the patient will be readmitted, the probability that they will actually be readmitted is .80
- If the patient is not readmitted, the probability of the model telling me so is .94
- If the patient is readmitted, the probability of the model telling me so is .23
- Confusion matrix indicates that of the 2001 data points used for testing, there were 974 true negatives, 166 true positives, 26 false positives, and 835 false negatives.

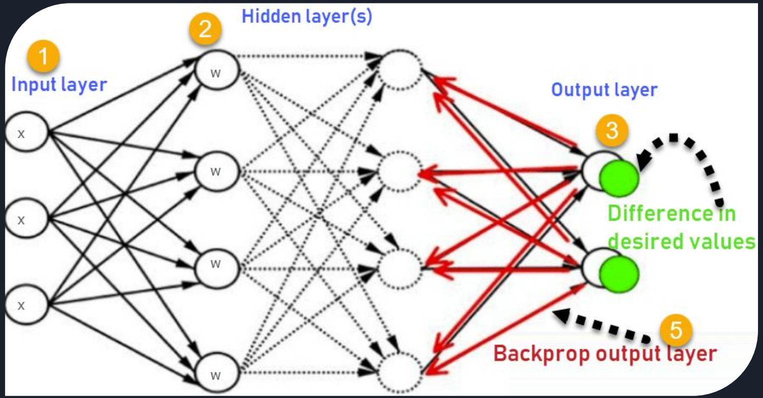
- ROC curve is not far off from the diagonal, indicating that the classifier is not much better than randomly guessing.
- Area under curve is .58538, but we want this number to be as close to 1 as possible.
- We need to tune the model!



Network Tuning

https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

- What factors improve a model?
- Deeper networks tend to perform better
    - Increase number of layers
    - Adjust number of nodes per layer
    - Try various activation functions
- Adjust number of epochs
    - Entire training set passed through network
    - Weights all updated once per epoch
    - Be careful about overfitting to training set
- Iterative experimentation process



Tuned Model

```
model = Sequential()
model.add(Dense(30, input_dim=n, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(8, activation='sigmoid'))
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=200, batch_size=10)
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy * 100))
```

Increased number of layers and nodes

Variation in activation functions

Epochs increased to 200

(Function for plotting ROC)

```
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()
```

Tuned Model : Evaluation Metrics

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score
from keras.models import load_model
```

Can view node weights

Model summary shows number of layers and nodes

```
    print(model.get_weights())
    print(type(model.get_weights()))
    print("Model Summary:")
    print(model.summary())
    scores = model.evaluate(X, y, verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1],
scores[1] * 100))
```

Save model, so you do not need to regenerate it every time

```
    # save model and architecture to file
    model.save("modeltuned.h5")
    print("Saved model to disk")

    print("loading model")
    # load model
    model = load_model('modeltuned.h5')
```

Input test data and get predictions from the model.

Print for comparison.

```
    # make class predictions with the model
    pred = model.predict_classes(X_test)
    for i in range(0, len(pred)):
        print('%d (expected %d)' % (pred[i],
y_test[i]))
```

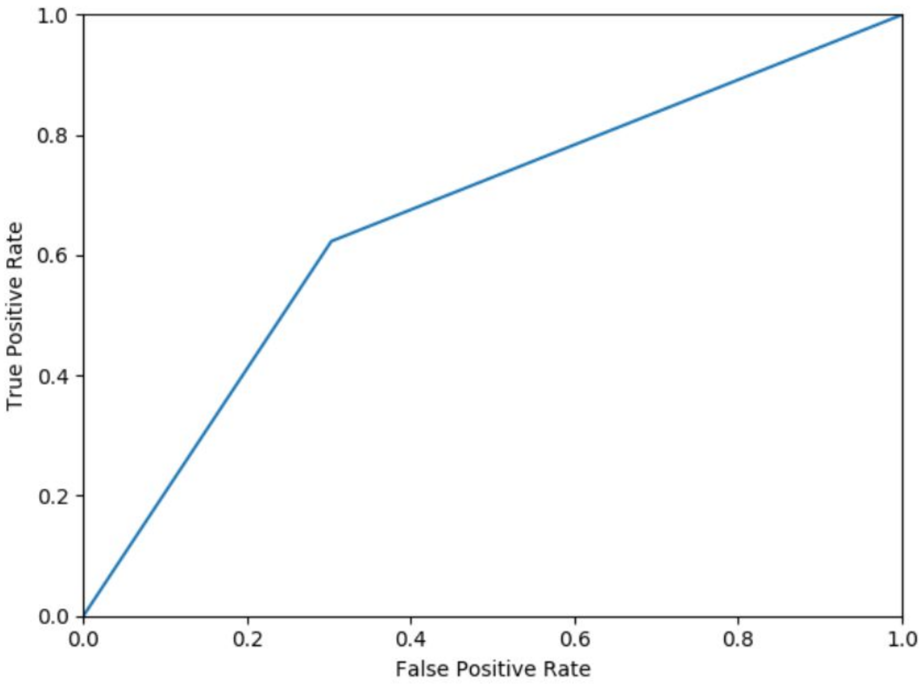
Calculate false positive and true positive rates for plotting ROC and getting AUC score

```
    fpr, tpr, thresholds = roc_curve(y_test, pred)
    plot_roc_curve(fpr, tpr)
    print("auc score")
    auc_score = roc_auc_score(y_test, pred)
    print(auc_score)
```

Get precision, recall, f1-score, and confusion matrix.

```
    print(classification_report(y_test, pred))
    print("confusion matrix:")
    print(confusion_matrix(y_test, pred))
```

Tuned Model Evaluation Statistics



Accuracy: 67.28%
AUC score = 0.6601883116883117

Classification Report

	precision	recall	f1-score	support
0.0	0.65	0.70	0.67	1000
1.0	0.67	0.62	0.65	1001
accuracy			0.66	2001
macro avg	0.66	0.66	0.66	2001
weighted avg	0.66	0.66	0.66	2001

Confusion Matrix:

[[697 303]
 [377 624]]

Interpretation

Accuracy: 67.28%
AUC score = 0.6601883116883117

Classification Report

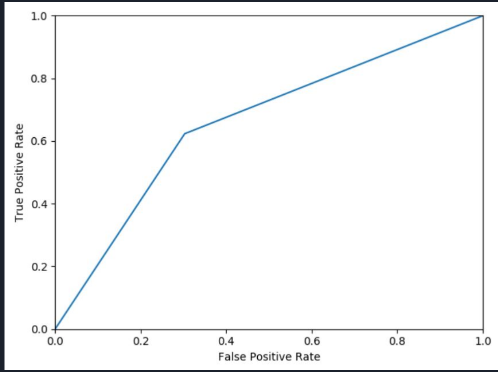
	precision	recall	f1-score	support
0.0	0.65	0.70	0.67	1000
1.0	0.67	0.62	0.65	1001
accuracy			0.66	2001
macro avg	0.66	0.66	0.66	2001
weighted avg	0.66	0.66	0.66	2001

Confusion Matrix:

[[697 303]
 [377 624]]

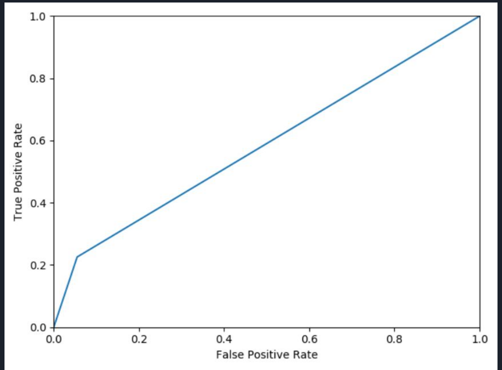
- If the model says the patient will not be readmitted, probability that they will actually not be readmitted is .65
- If the model says the patient will be readmitted, the probability that they will actually be readmitted is .67
- If the patient is not readmitted, the probability of the model telling me so is .70
- If the patient is readmitted, the probability of the model telling me so is .62
- Confusion matrix indicates that of the 2001 data points used for testing, there were 697 true negatives, 624 true positives, 303 false positives, and 377 false negatives.

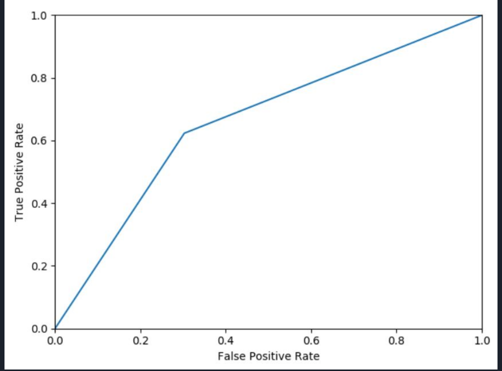
- ROC curve is farther from the diagonal, indicating that the classifier has more reliable predictive power.
- Area under curve is .66018, which is closer to 1 than before



Comparison of Base and Tuned Models

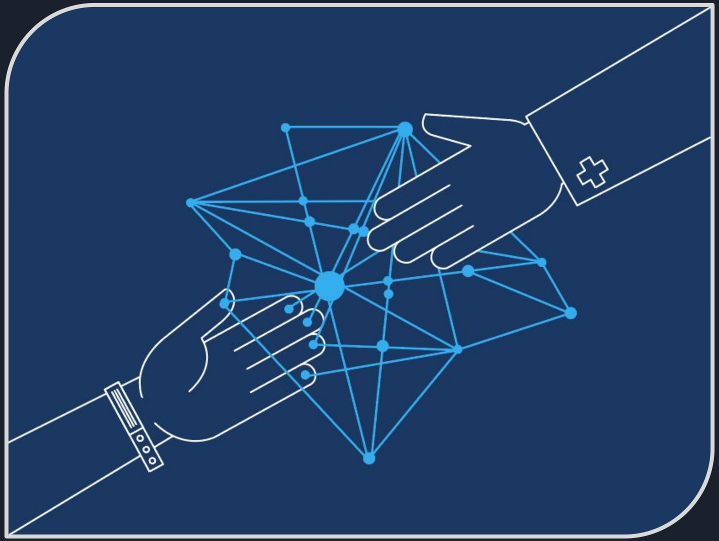
- ROC shows second model is better at discerning between positive and negative classes
  - Area increased from .58 to .66
- Accuracy of predictions on the same testing data set increased from 60.02% to 67.28%.
- Proportion of true positive and true negative predictions are about the same in second model
  - Initial model had skewed results (especially inaccurate in predicting positive cases).





- F1 score combines precision and recall (harmonic mean), and a model should maximize this to have an optimal balance of these two metrics. Should be as close to 1 as possible.
  - Original model f1 scores: .69 for classification 0 & .35 for classification 1
  - Improved model f1 scores: .67 for classification 0 & .65 for classification 1.
- F1 score should be close to 1 for both classes to indicate a better classifier--confirms second model as more reliable predictive model.

Can we predict a patient's likelihood of hospital readmission?



- This model indicates that there may be a correlation between a patient's lifestyle and chances of being readmitted.
- Accuracy is not high enough to be definitive, but a model may be a helpful tool in tandem with a professional's advice.
- However, neural network is more accurate in this case than regression models were, which is promising.
- With more exhaustive tuning of this model, there is a prospect of creating a model that is increasingly reliable in providing predictions.

Concluding Thoughts