Laboratory Session 5

Random Forest –KNN-Neural Network

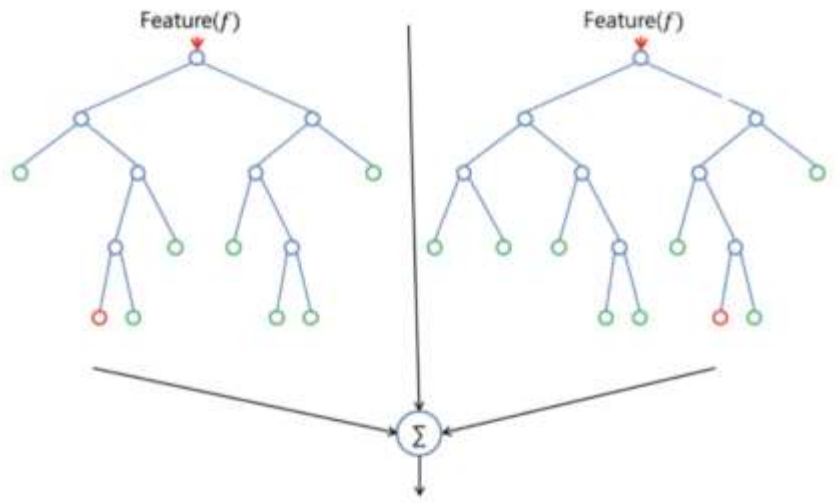
1. Random Forest

Tutorial links: 1. Definition: [https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd]; 2. Random Forest algorithm [https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/] 3. Random Forest in Python: [https://towardsdatascience.com/random-forest-in-python-24d0893d51c0] or [https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html]

- Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyperparameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks.

1.1 How it works:

Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The "forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.



- Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

- Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

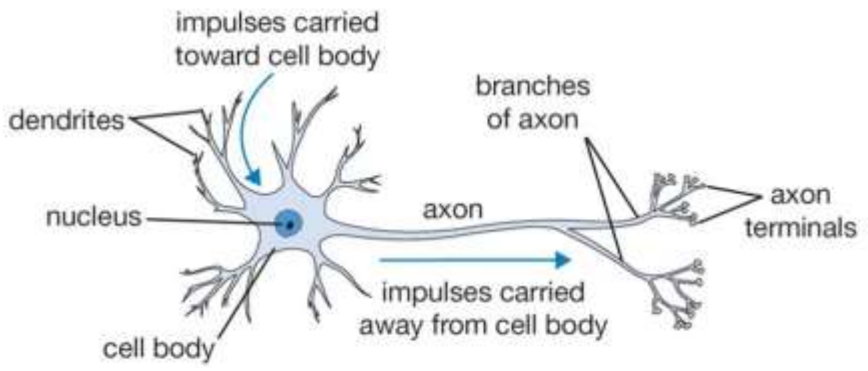
1.2 Feature Importance:

- Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the **sum of all importance is equal to 1.**
- Through looking at the feature importance, you can decide which features you may want to drop, because they don't contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa.

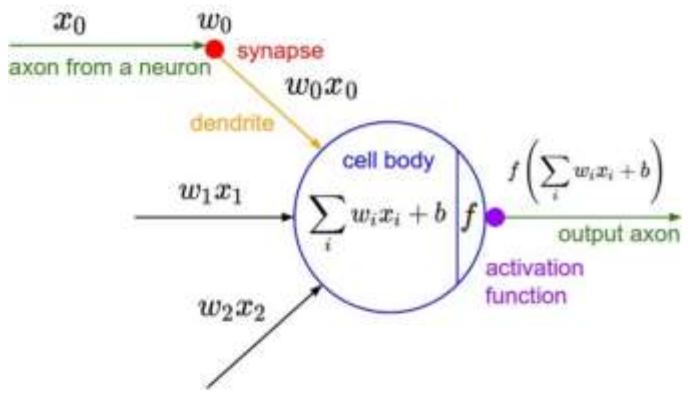
2. Neural Network

Tutorial links: 1. **Step by step** [https://www.datacamp.com/community/tutorials/deep-learning-python?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=s&utm_adpostion=1t1&utm_creative=332602034358&utm_targetid=aud-299261629574:dsa-473406573755&utm_loc_interest_ms=&utm_loc_physical_ms=1028581&gclid=CjwKCAjwza_mBRBTEiwASDWVvgqqM6mD6uEWihqOU86on6m4NxvzODvgS9m9pq3QEbJfcIO7PAmipRoCtuUQAvD_BwE] or [https://medium.com/analytics-vidhya/neural-networks-for-digits-recognition-e11d9dff00d5
2. Video1: **Forward propagation**: https://campus.datacamp.com/courses/deep-learning-in-python/basics-of-deep-learning-and-neural-networks?ex=3
3. Video2: **Activation functions**: https://campus.datacamp.com/courses/deep-learning-in-python/basics-of-deep-learning-and-neural-networks?ex=5
4. Video3: **Deeper networks**: https://campus.datacamp.com/courses/deep-learning-in-python/basics-of-deep-learning-and-neural-networks?ex=8

- Neural networks are a biologically-inspired algorithm that attempt to mimic the functions of neurons in the brain. Each neuron acts as a computational unit, accepting input from the dendrites and outputting signal through the axon terminals. Actions are triggered when a specific combination of neurons are activated.


impulses carried toward cell body
branches of axon
dendrites
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

- The neural network is usually designed to output a vector containing probabilities of the data being in one of the possible categories



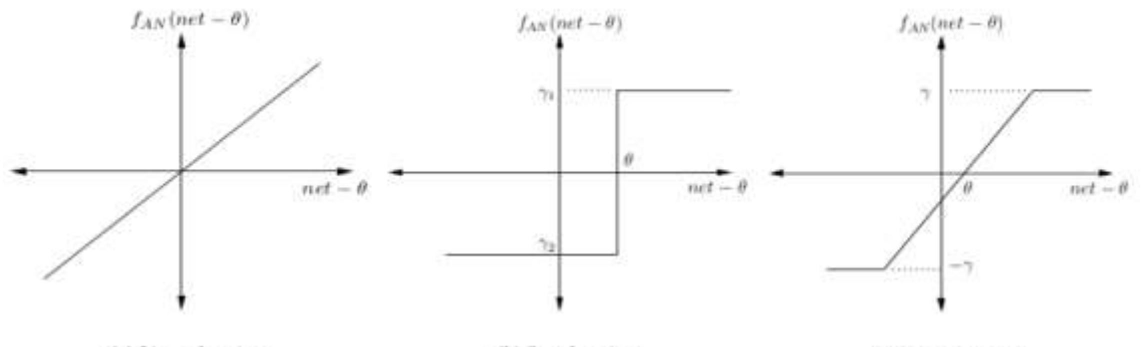
For example, in the case of recognizing whether an image contains a dog or a cat, the neural network could output a vector [0.3, 0.7]. If the network is trained in such a way that the first value in the output vector represents the probability that the image contains a dog and the second value represents the probability that the image contains a cat, then the network is 70% certain that the image contains a cat. The network is trained data which is segmented into two pieces – the data itself, and the labels.

- **Sigmoid** is the first type of output unit, and activation unit, most people learn. It is very simple, and it used to be used much more. A sigmoid unit is used when predicting the value of a binary variable. That means it can predict for only two cases

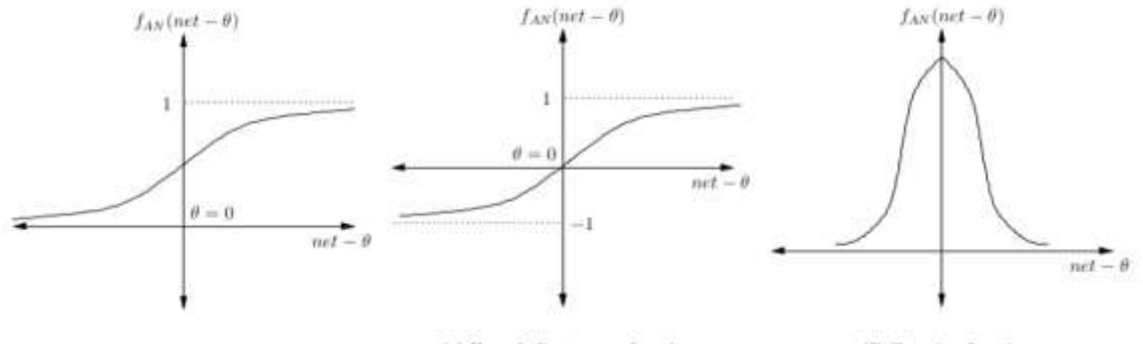
$$\sigma = \frac{1}{1+e^{-x}} \quad \rightarrow \quad y = \sigma(w^T h + b)$$

- **Softmax** unit is used when we want to the probability distribution over a discrete number of possible outcomes. The output vector of the softmax layer contains values in the interval [0, 1]. Just like with the sigmoid function, each number represents the percentage of probability. The sum of all values in the output vector must always equal 1.

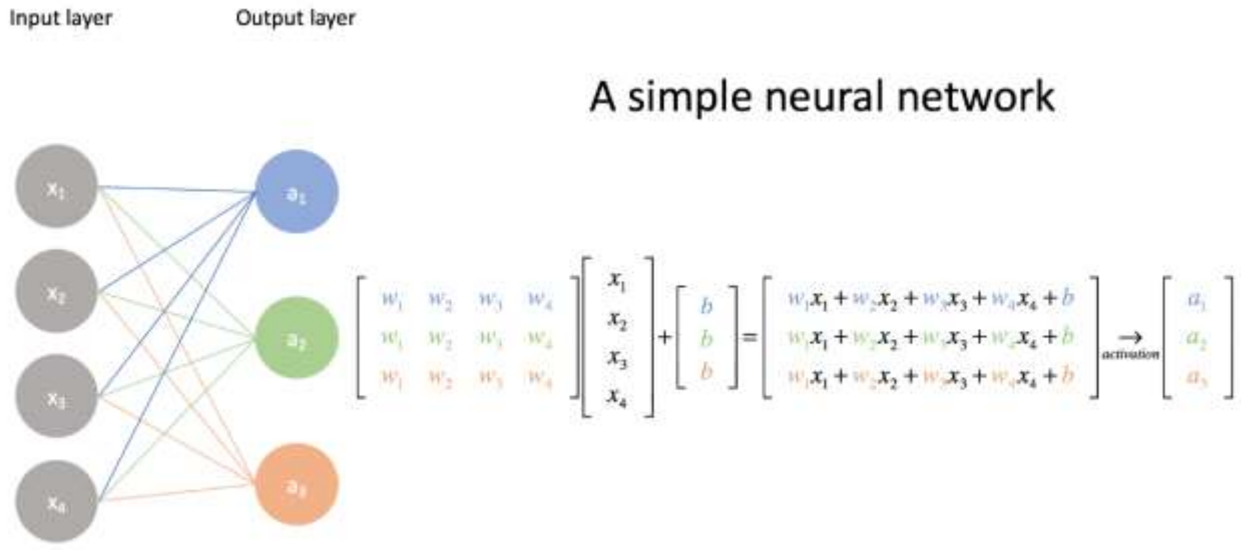
$$z = W^T h + b \;\rightarrow\; softmax(z)_i = \frac{e^{z_i}}{\Sigma_j e^{z_j}}$$



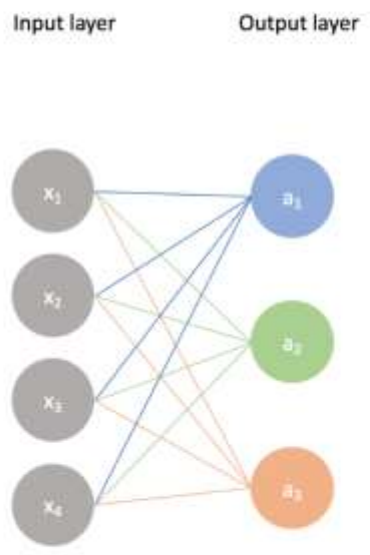
(a) Linear function (b) Step function (c) Ramp function

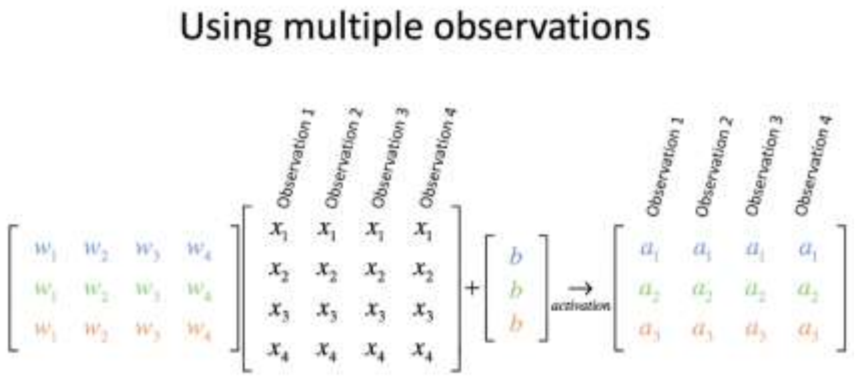


(d) Sigmoid function (e) Hyperbolic tangent function (f) Gaussian function

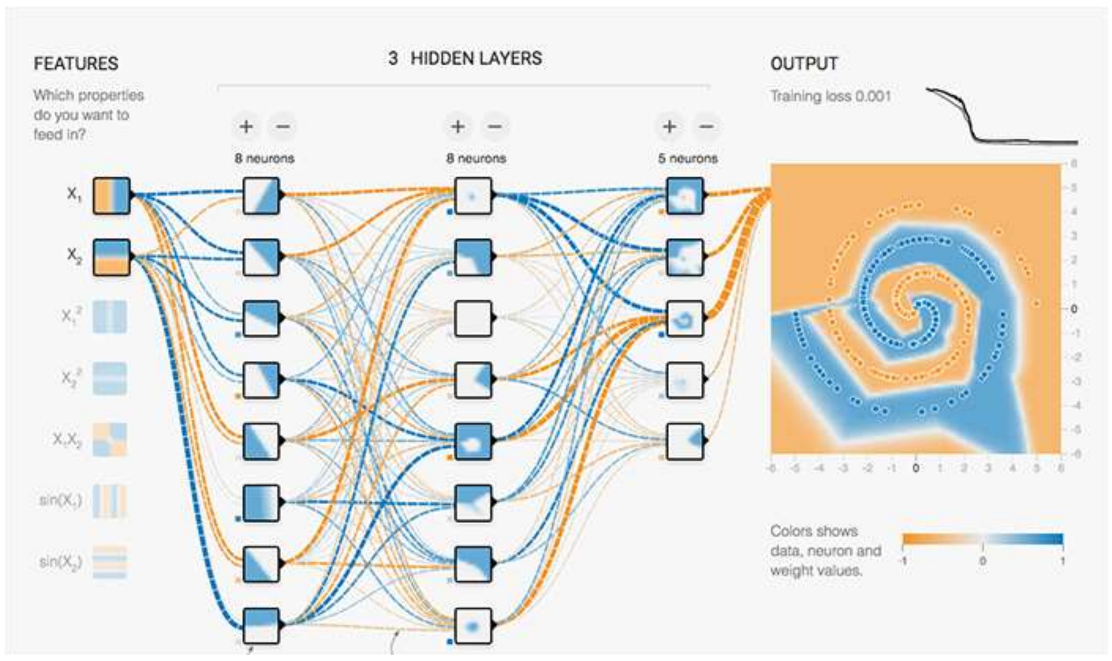


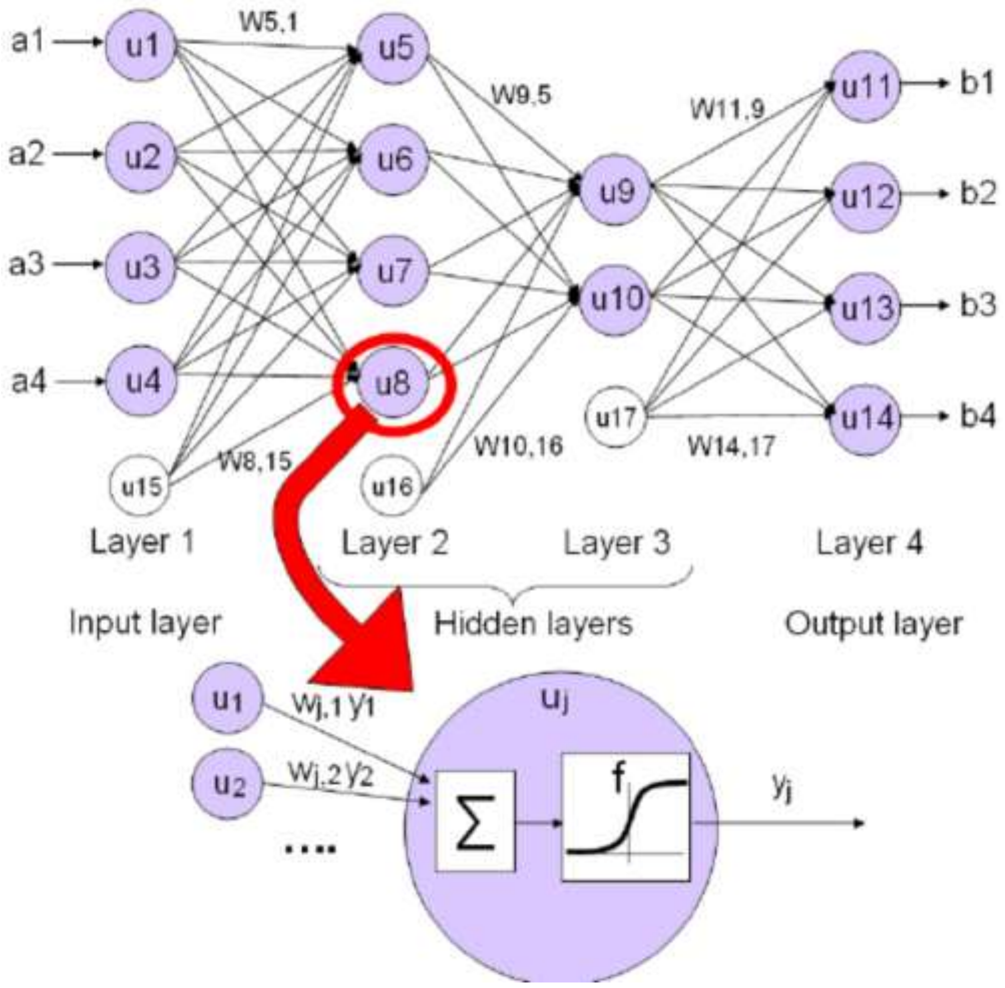
$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[\text{activation}]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$



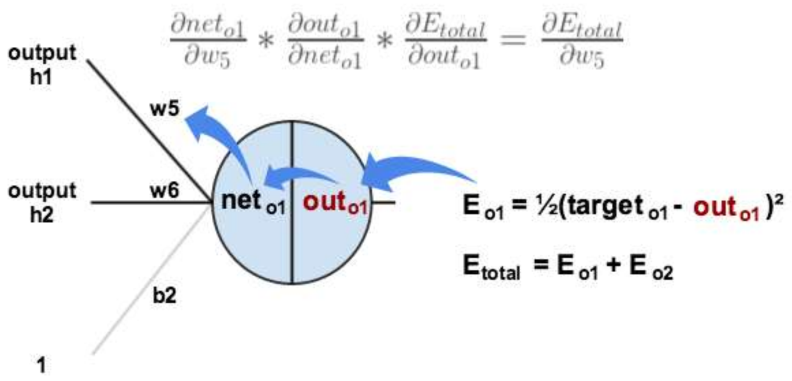


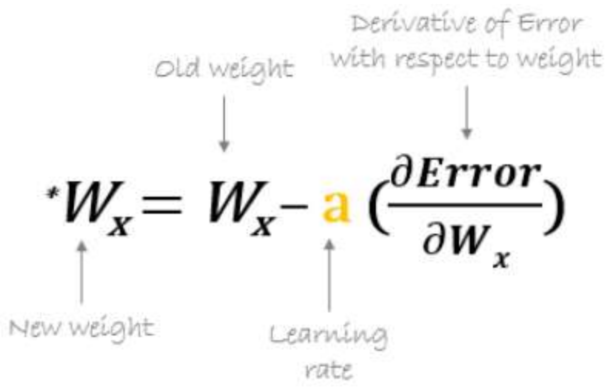
- **Hidden layers:**
    - The real power of neural networks emerges as we add additional layers to the network. Any layer that is between the input and output layers is known as a **hidden layer**. Thus, the following example is a neural network with an input layer, one hidden layer, and an output layer.


FEATURES
Which properties do you want to feed in?
3 HIDDEN LAYERS
8 neurons
8 neurons
5 neurons
OUTPUT
Training loss 0.001
Colors shows data, neuron and weight values.



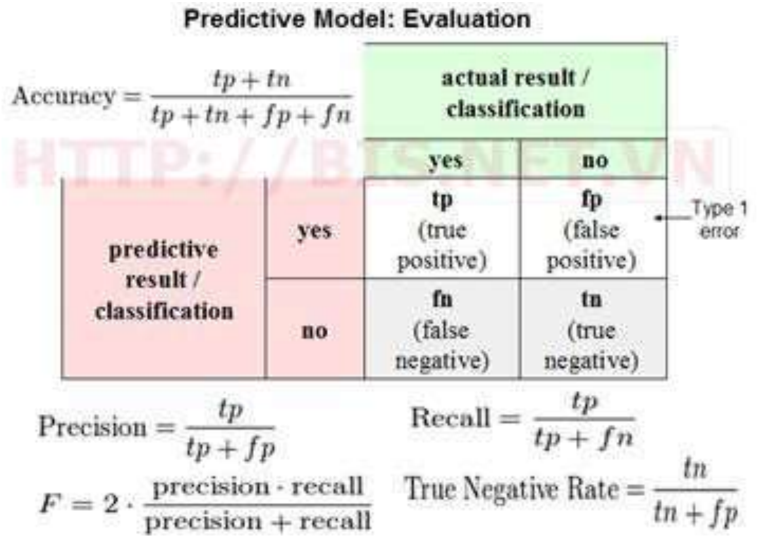
- **Back Propagation:**
    - Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network's layers. It is commonly used to train deep neural networks.


$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$
output h1
output h2
w5
w6
b2
1
net o1
out o1
$E_{o1} = ½(target_{o1} - out_{o1})^2$
$E_{total} = E_{o1} + E_{o2}$


Derivative of Error with respect to weight
Old weight
$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$
New weight
Learning rate

Performance Metrics
Confusion Matrix: It is nothing but a tabular representation of Actual vs Predicted values. This helps us to find the accuracy of the model and avoid over-fitting. -The confusion matrix, which is a breakdown of predictions into a table showing correct predictions and the types of incorrect predictions made. Ideally, you will only see numbers in the diagonal, which means that all your predictions were correct!
-Precision is a measure of a classifier's exactness. The higher the precision, the more accurate the classifier.
-Recall is a measure of a classifier's completeness. The higher the recall, the more cases the classifier covers.
-The F1 Score or F-score is a weighted average of precision and recall.
-The Kappa or Cohen's kappa is the classification accuracy normalized by the imbalance of the classes in the data.

-



Predictive Model: Evaluation

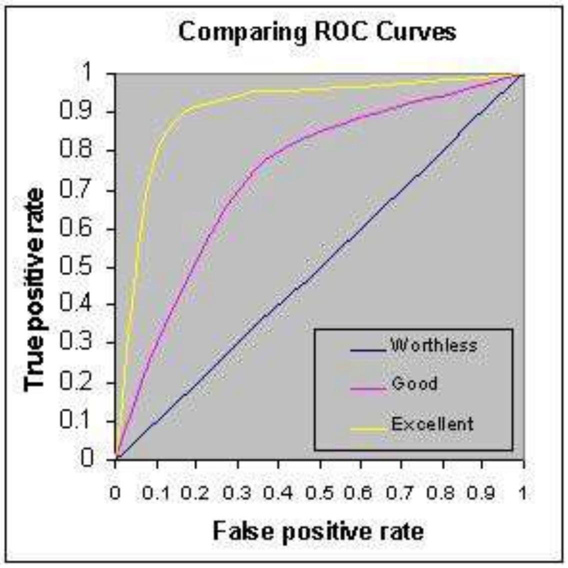
$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

		actual result / classification	
		yes	**no**
predictive result / classification	**yes**	**tp** (true positive)	**fp** (false positive) ← Type 1 error
	no	**fn** (false negative)	**tn** (true negative)

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \qquad \text{True Negative Rate} = \frac{tn}{tn + fp}$$

- **ROC Curve:** Receiver Operating Characteristic (ROC) summarizes the model's performance by evaluating the trade-offs between true positive rate (sensitivity) and false positive rate (1- specificity). For plotting ROC, it is advisable to assume p > 0.5 since we are more concerned about success rate. ROC summarizes the predictive power for all possible values of p > 0.5. The area under curve (AUC), referred to as index of accuracy (A) or concordance index, is a perfect performance metric for ROC curve. Higher the area under curve, better the prediction power of the model. Below is a sample ROC curve. The ROC of a perfect predictive model has TP equals 1 and FP equals 0. This curve will touch the top left corner of the graph.



3. Practice

-read data 'CustomerChurn.csv'

-Import machine learning libraries

```
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import linear_model, datasets
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC, SVC
```

-Split data and train data with encoder set similar to **Lab 3 (or Lab 4)**

3.1 Random Forest

-define the function **def RandomForestLearning(DataTrain, TargetTrain):**

from sklearn.ensemble import RandomForestClassifier

#type your code

return RF

- define the function **def RandomForestTesting(RFModel, DataTest, TargetTest):**

from sklearn.metrics import accurace_score

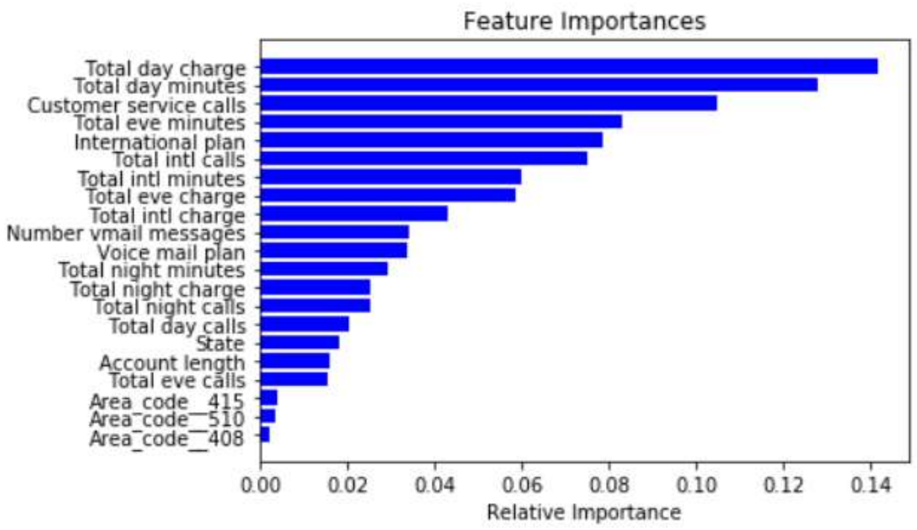
#type your code

return AccuracyRF, PredictTestRF

- Learn RF method (without feature importance) and print out the accuracy and running time.

-Find the feature and importance as follows:

+import RandomForestRegressor (from sklearn.ensemble import RandomForestRegressor)
+create model of RandomForestRegreesor with random_state=1 and max_depth=10
+fit mode with data_train_encoder and target_train_encoder
+create features with data_train_encoder.columns
+sort features ascending
+plot figure of RF as follows



-Get numerical feature_list and importance_list as follows [Hint using list() and zip() [https://www.geeksforgeeks.org/zip-in-python/] to list of tuples with variable and importance as follows

-Sort the feature importance by most importance first

-And print out the feature and importance

```
Variable: Total day charge     Importance: 0.14
Variable: Total day minutes    Importance: 0.13
Variable: Customer service calls Importance: 0.1
Variable: International plan   Importance: 0.08
Variable: Total eve minutes    Importance: 0.08
Variable: Total intl calls     Importance: 0.08
Variable: Total eve charge     Importance: 0.06
Variable: Total intl minutes   Importance: 0.06
Variable: Total intl charge    Importance: 0.04
Variable: Voice mail plan      Importance: 0.03
Variable: Number vmail messages Importance: 0.03
Variable: Total night minutes  Importance: 0.03
Variable: Total night calls    Importance: 0.03
Variable: Total night charge   Importance: 0.03
Variable: State                Importance: 0.02
Variable: Account length       Importance: 0.02
Variable: Total day calls      Importance: 0.02
Variable: Total eve calls      Importance: 0.02
Variable: Area_code__408       Importance: 0.0
Variable: Area_code__415       Importance: 0.0
Variable: Area_code__510       Importance: 0.0
```

# Split Train and Test and check shape

-AttSelection = ["Total day charge", "Total day minutes", "Customer service calls", "International plan", "Total eve minutes", "Total intl calls", "Total eve charge", "Total intl minutes", "Total intl charge", "Voice mail plan", "Number vmail messages", "Total night minutes", "Total night calls", "Total night charge", "Churn"]

-data_train_encoder_feselection02,target_train_encoder_feselection02, -data_test_encoder_feselection02, target_test_encoder_feselection02 = SplitDataFrameToTrainAndTest(data_encoder[AttSelection], 0.6, 'Churn')
-PrintTrainTestInformation(data_train_encoder_feselection02, target_train_encoder_feselection02, data_test_encoder_feselection02, target_test_encoder_feselection02)

- Learn RF method with feature importance and print out the accuracy and running time.

3.2 KNN

--define the function **def KNNLearning(DataTrain, TargetTrain):**

```
        from sklearn.neighbors import KNeighborsClassifier
        #type your code
      return KNN
```

- define the function **def KNNTesting(RFModel, DataTest, TargetTest):**

```
        from sklearn.metrics import accurace_score
        #type your code
      return AccuracyKNN, PredictTestKNN
```

-**Learn KNN** method (with and without feature importance) and print out the accuracy and running time.

3.2 Neural Network

Tutorial link: step by step [https://www.datacamp.com/community/tutorials/deep-learning-python?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=s&utm_adpostion=1t1&utm_creative=332602034358&utm_targetid=aud-299261629574:dsa-473406573755&utm_loc_interest_ms=&utm_loc_physical_ms=1028581&gclid=CjwKCAjwza_mBRBTEiwASDWVvgqqM6mD6uEWihqOU86on6m4NxvzODvgS9m9pq3QEbJfcIO7PAmipRoCtuUQAvD_BwE]

-import Sequential for models (from keras.models import Sequential)
-Import Dense for layers (from keras.layers import Dense)
-Import ModelCheckpoint for callbacks (from keras.callbacks import ModelCheckpoint)
-create random.seed as follows

```
seed = 42
np.random.seed(seed)
```

- create the model by passing a list of layer instances to the constructor, which you set up by running model = Sequential()

-Add input layer (1st layer): input_dim=21, 12 nodes, init='uniform' and activation='relu'. Hint using model.add()

-Add one hidden layer (2nd layer): 8 nodes, init='uniform' and activation ='relu'. Hint using model.add()

-Add one output layer (2nd layer): 1 nodes, init='uniform' and activation ='relu'. Hint using model.add()

-Report model output shape, model summary, model config and list all weight tensors.
-Retest all traditional classification approaches with:

+num of epochs to test for: NB_EPOCHS=500
+BATCH_SIZE=15
+ checkpoint: store the best model: 'pima-weights.best.hdf5'
+train the model, store the results for plotting. Hint: using model.fit with verbose=0 and callbacks_list is checkpoint

```
Epoch 00018: val_acc did not improve from 0.90698

Epoch 00019: val_acc did not improve from 0.90698

Epoch 00020: val_acc did not improve from 0.90698

Epoch 00021: val_acc improved from 0.90698 to 0.90848, saving model to pima-weights.best.hdf5

Epoch 00022: val_acc did not improve from 0.90848
```

-Please evaluate basic classification techniques, such as:

+ confusion matrix,
+Precision
+Recall
+ F1 Score or F-score
+Kappa or Cohen's kappa

-Please draw ROC curve

4. Please make conclusions about Accuracy, Running Time, Figures of all methods.