ASSIGNMENT 05 DISCUSSION REQUIRED TO UNDERSTAND PROVIDED CODE SAMPLES

Coding Track Discussion 03

Bhaskarjyoti Das
19/03/2018

WHY THIS ?

1. In this assignment we need to be comfortable plugging in text data into an ML application.
2. We have provided helper code that you may reuse for this assignment. The notebooks do not have explanations. This material gives necessary background
3. This is also to address some of the “unknowns” some of you will face in your projects while marrying NLP with ML as many are doing projects as NLP+ML .

WHAT WE ASKED YOU TO DO

1. To fit text data into a categorization application (using the ML models that you have learnt in your ML course)
2. Fitting a model is relatively easy (using standard API) but cleaning text and deriving features are the hard parts!
3. You need to do a linear (example - SVM) and a non-linear (example - ANN) model in the five cases.
4. Compare the 10 cases using metrics (each has 2 sub cases of modeling)
   a) Tf-idf vectors from text (limit vocabulary: 25k -> 15K ?)
   b) Since even after limiting, the vocab can be large, do dimensionality reduction (use SVD preferably ensuring that you capture >90% variances. 15k -> 2K ?)
   c) Derive features (such as emotions and sentiment) using lexical semantics
   d) Derive features (vocabulary -> emotion carrying words)
   e) Pure Distributional semantics (word2Vec)

KIND OF CODE THAT YOU WILL FIND IN G DRIVE

1. Template code to calculate sentiment from text data using lexical semantics / trained classifier
2. Template code to get emotion(s) from text data using lexical semantics
3. Template code to create Word2Vec word embedding from text data such as reviews.
4. Template code to use Doc2Vec word embedding
5. template code for ML using scikit and keras for non text data, calculating metrics
6. Template code to convert text data into vectorised form

WHAT CODE IS TO BE WRITTEN THEN ?

1. Most of the code that needs time to test/write are given in some template form to save time
2. Students can write their own implementation essentially by gluing all these pieces. This is to save their time.
3. Trivial code is not given. For example, code to extract the all the review texts into a single text file to create the word2vec embedding for the reviews in the dataset is not provided.

THE DATASET

1. It has 10000 rows and 10 columns. You are interested mostly in “text” and “stars”.
2. “stars” have the classes. There are 5 of them.
3. You can simplify your problem by opting to work only with 1,5 or 1,2-> 0 and 4,5 -> 1. You also should check that the data is kind of balanced (imbalanced classes)
4. You need some pre-processing of the text even if you just use tf-idf as vectorization method.

SCIKIT LEARN API : ESTIMATOR

All objects share a consistent interface

Estimator

- any object that can estimate some parameter based on a dataset is an estimator. Example – imputer that estimates missing value based on some strategy
- Estimation itself is performed by fit() i.e. takes a dataset and a set of labels as possible parameters
- Any other hyper parameter (strategy for imputer for example) must be an instance variable

SCIKIT LEARN API : TRANSFORMER

Transformer

- Some estimators can transform a dataset (example – imputer) using transform(). There is a convenience method called fit_transform() that is same as calling fit() and then transform()
- Fitting finds the internal parameters of a model that will be used to transform data. Transforming applies the parameters to data.
- You may fit a model to one set of data, and then transform it on a completely different set.
- Transformer can be thought of as a data in, data out black box. Generally, they accept a matrix as input and return a matrix of the same shape as output. That makes it easy to reorder and remix them at will. MinMaxScaler is a built in transformer

SCIKIT LEARN API : PREDICTOR

Predictors

- Some estimators are capable of making predictions against a dataset . For example, LinearRegression().
- It typically has a score() method that gives the quality of prediction

Inspection of what is going on

- All estimator's hyperparameters are available via public instance variables. Example – imputer.strategy
- All estimators learned parameters are available via public instance variable with a _ suffix i.e. imputer.statistics_

SCIKIT LEARN API : HYPER PARAMETERS

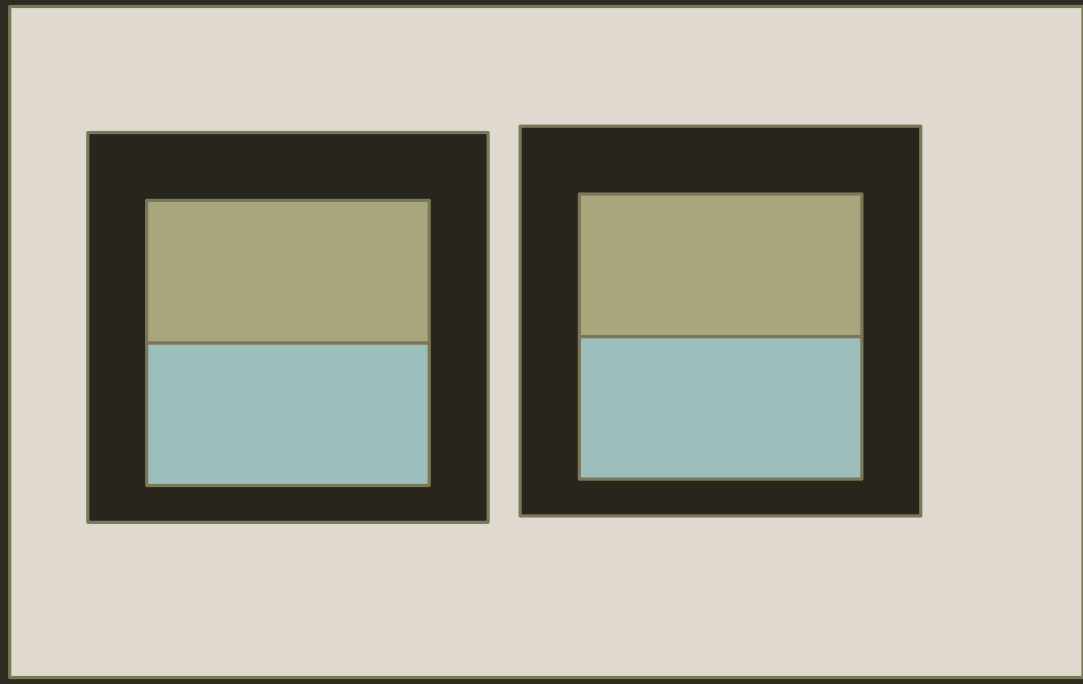
Hyperparameters are Python strings or numbers

Dataset are Scipy Sparse matrix or Numpy array

There are sensible default values for most parameters

SCIKIT LEARN API : PIPELINES

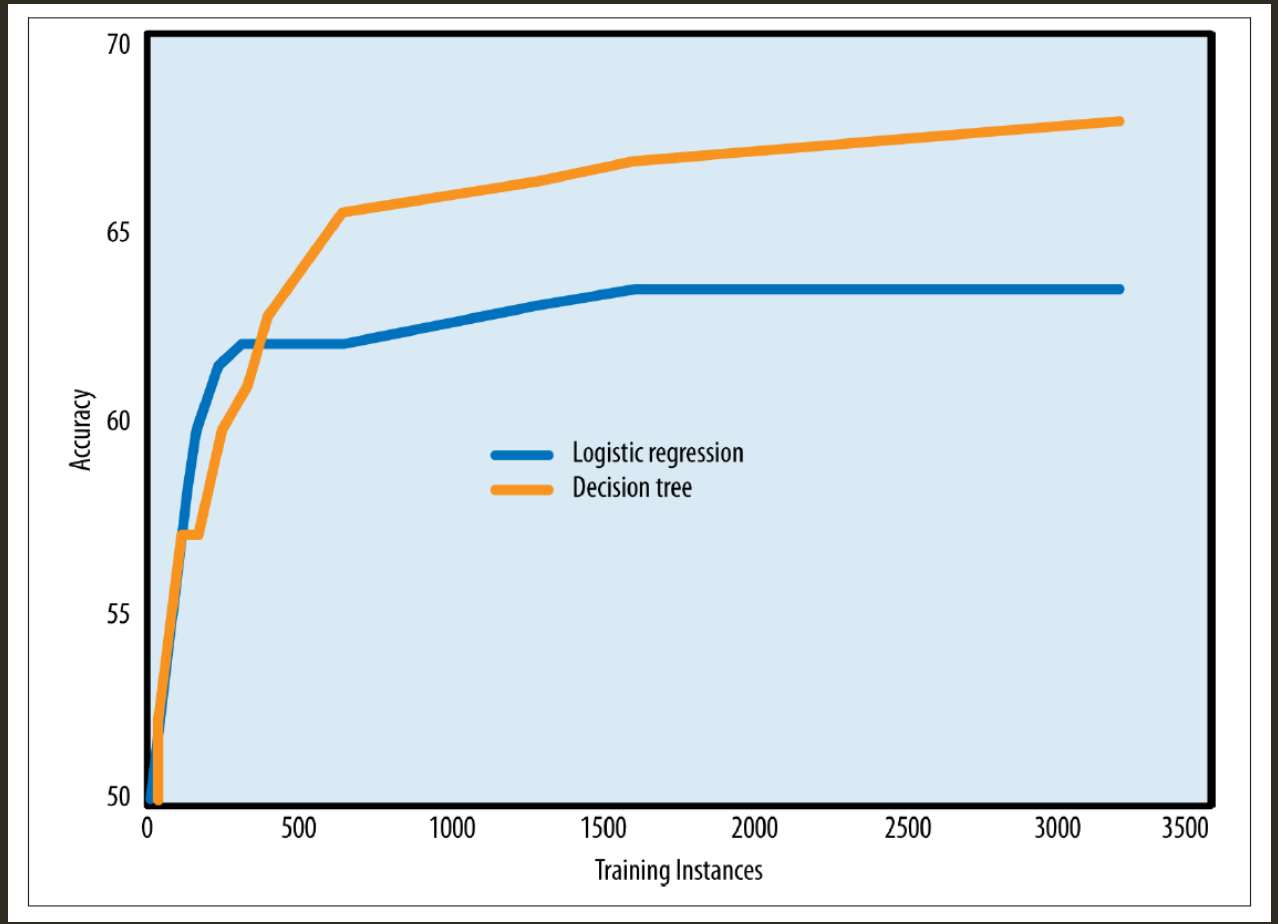
1. Pipelines are **container of steps** used to package a workflow
   - Estimators
   - Transformers
   - Pipeline
   - FeatureUnions
2. Pipelines are modular like Lego blocks. Example – in the picture, an estimator and a transformer are put together by a pipeline
3. FeatureUnion can combine the result of two pipelines



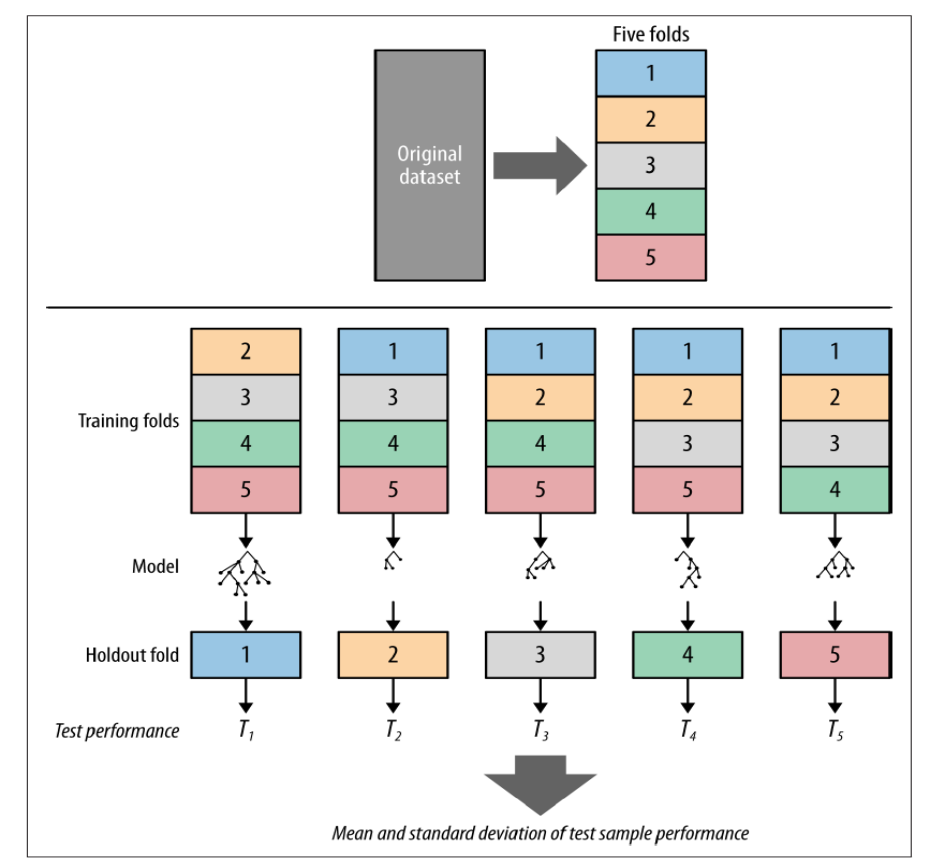
ADDRESS OVERFITTING (1) : GET AS MUCH DATA AS POSSIBLE

As size of data increases, generalization error decreases or accuracy of model increases but finally levels off

What next ?



ADDRESS OVERFITTING (2) GET THE MAX OUT OF DATA



K fold cross validation over

- Which **features**
- Which **models**
- Which **hyper parameters** of the model

ADDRESS OVERFITTING (3) : USE ENSEMBLE LEARNING

Individual models tend to over fit

- If you train multiple models, overfitting aspects of each of the models would be different

When **we combine these models**

- Overfitting components of models tend to cancel each other
- Left with components that really describe the data

Models in an ensemble can be based on **different algorithms (bagging, boosting), different features, different training sets, different hyper parameters**

The output of the models **can be combined** (majority, average, bagging, boosting, stacking etc.)

Story : Netflix prize winners used more than 100 models in ensembles

ADDRESS OVERFITTING (4) : PENALIZE COMPLEXITY

Regularization is the term for it

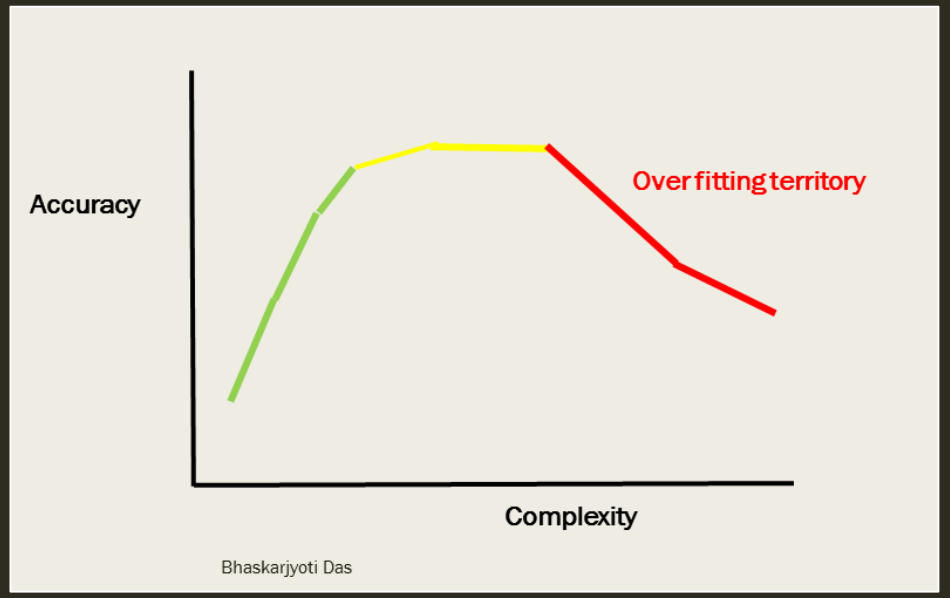
$E_{revised}(model) = E(model) + \lambda * R(model)$

- λ controls importance of **regularizing term**
- R (model) is the **regularizing term** that increases with complexity

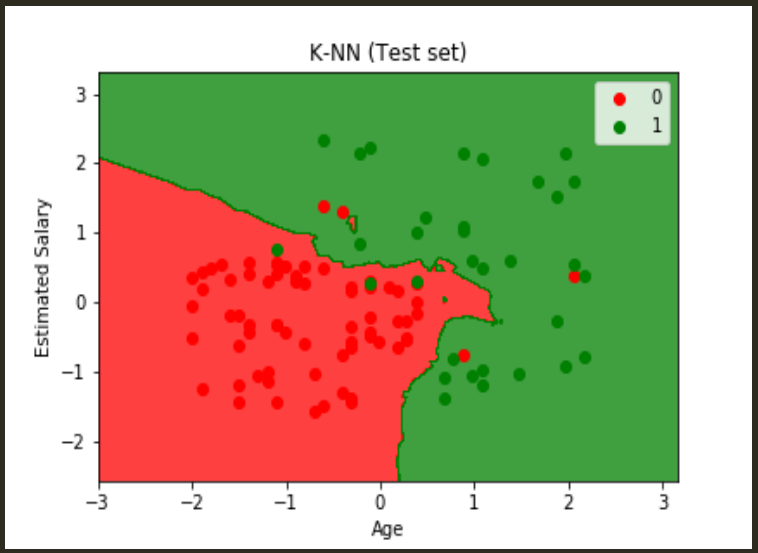
We get a model that gives **low error** and at the same times keeps the **complexity low** (by penalizing it).

Example: R-squared vs. Adjusted R-squared in regression

A library like ScikitLearn will give this option.

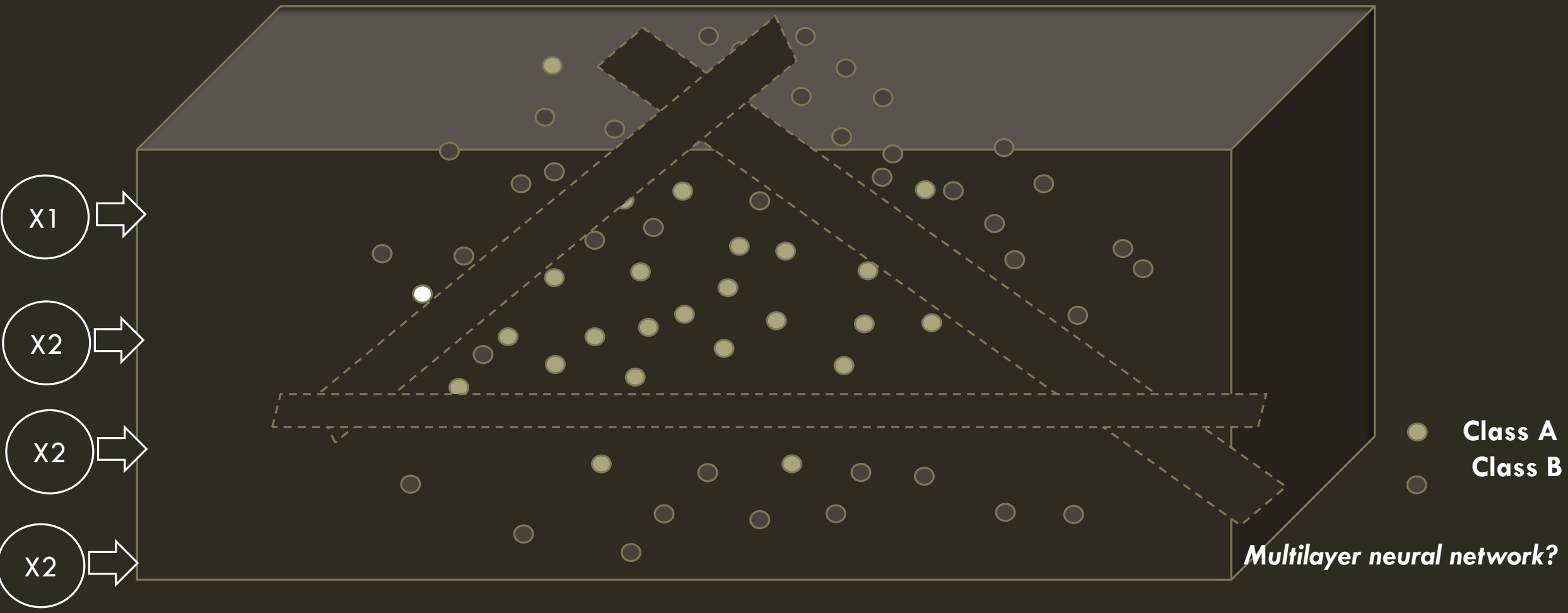


CLASSIFIERS DIFFER BY THEIR DECISION BOUNDARY



We tried out with same toy dataset …

DECISION BOUNDARY FOR MULTILAYER NEURAL NETWORK

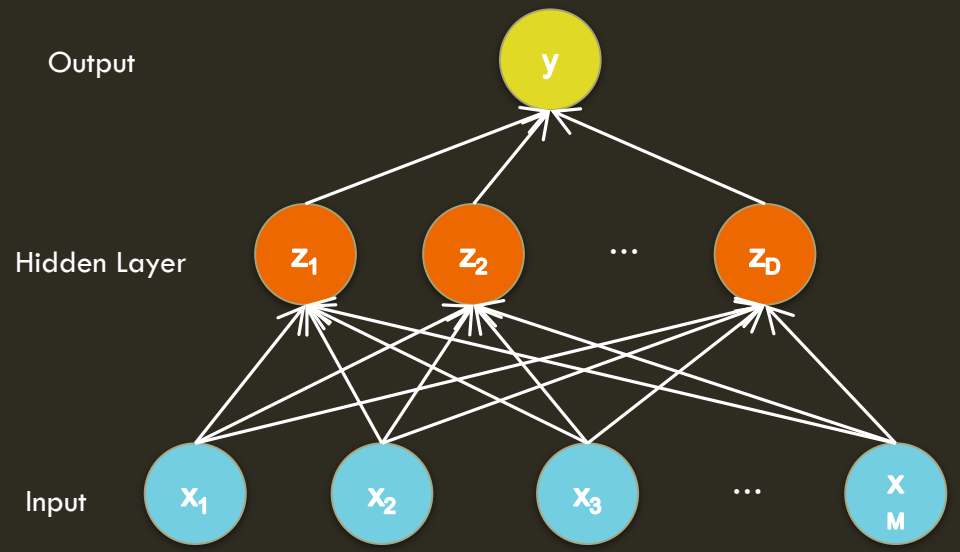


WHY SIGMOID MAY NOT BE THE BEST ACTIVATION FUNCTION ?

1. It takes a real-valued number and “squashes” it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1.
2. *Sigmoids saturate and kill gradients*. At either tail of 0 or 1, the gradient at these regions is almost zero. During backpropagation, this (local) gradient will be multiplied to the gradient of this gate’s output for the whole objective. Therefore, if the local gradient is very small, it will effectively “kill” the gradient and almost no signal will flow through the neuron to its weights and recursively to its data.

ACTIVATION FUNCTIONS : USUAL

Neural Network with arbitrary nonlinear activation functions



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (nonlinear)**
$y = \sigma(b)$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

SOFTMAX ACTIVATION FOR MULTICLASS

The **Softmax function**

- Sigmoid function can only handle two classes. For multi class classification, Is often used in the final layer.
- The Softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. **But it also divides each output such that the total sum of the outputs is equal to 1**
- If there are multiple classes and each input can belong to **exactly** one class, then it absolutely makes sense to use softmax activation in final layer.
- Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, ..., K.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

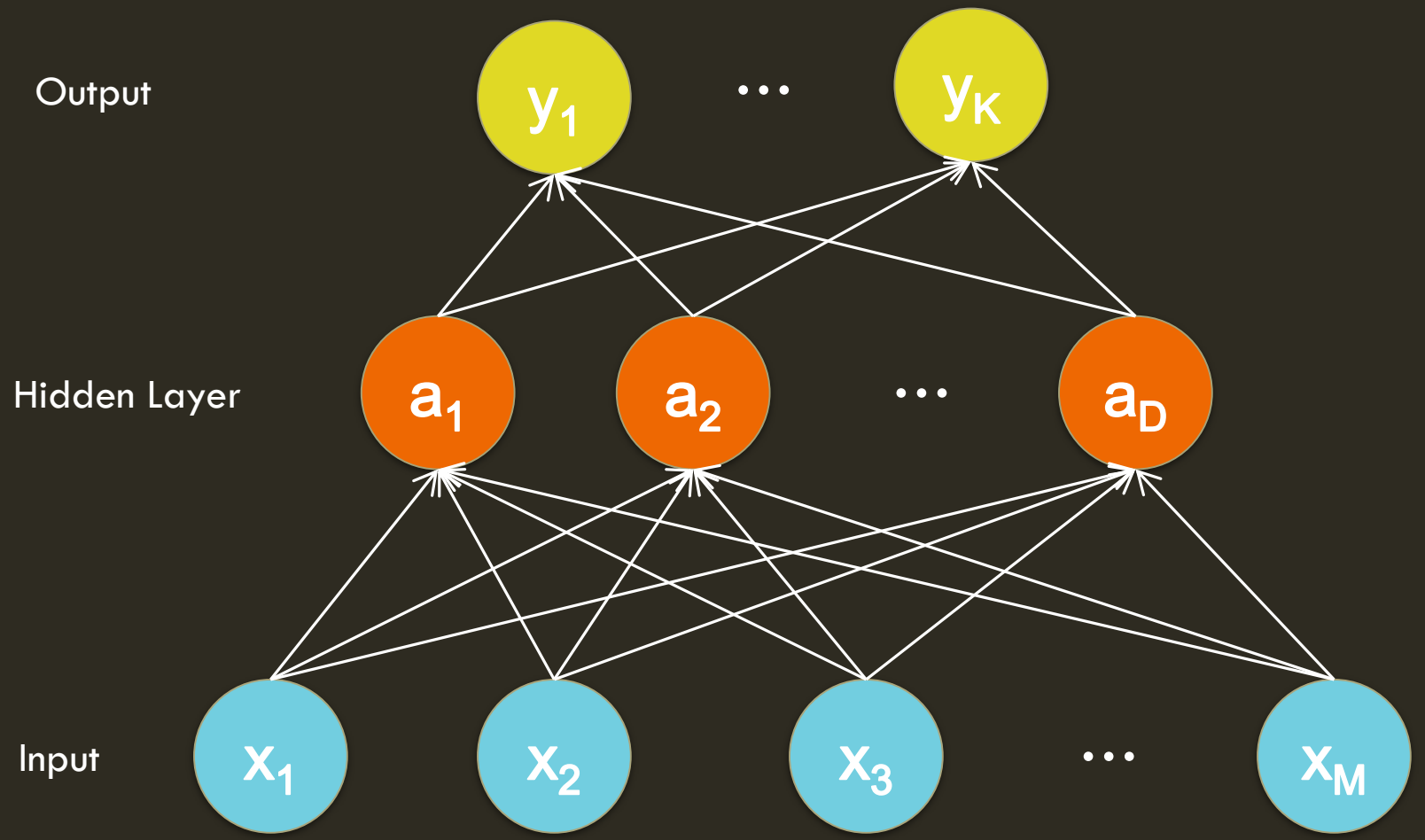
CROSS ENTROPY LOSS/ERROR FUNCTION FOR SOFTMAX ACTIVATION

1. Typically the loss function used with softmax is cross entropy

   $J = -1/N \left(\sum_{i=1}^{N} y_i \cdot \log(\check{Y}_i)\right)$

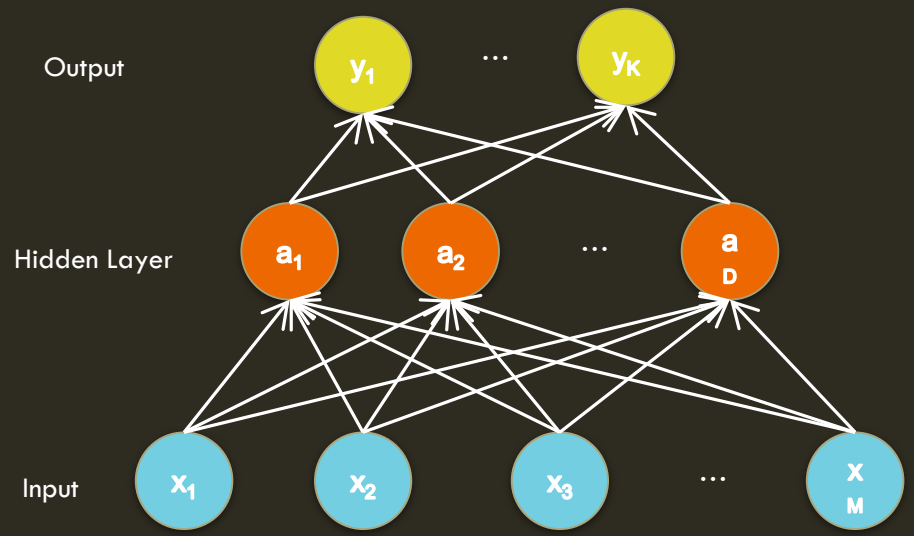
1. Notice that when computing mean cross entropy error with neural networks in situations where target outputs consist of a single 1 with the remaining values equal to 0, all the terms in the sum except one (the term with a 1 target) will vanish because of the multiplication by the 0s.
2. Put another way, cross entropy essentially ignores all computed outputs which don't correspond to a 1 target output. The idea is that **when computing error during training,** you really don't care how far off the outputs which are associated with non-1 targets are, you're only concerned with how close the single computed output that corresponds to the target value of 1 is to that value of 1.

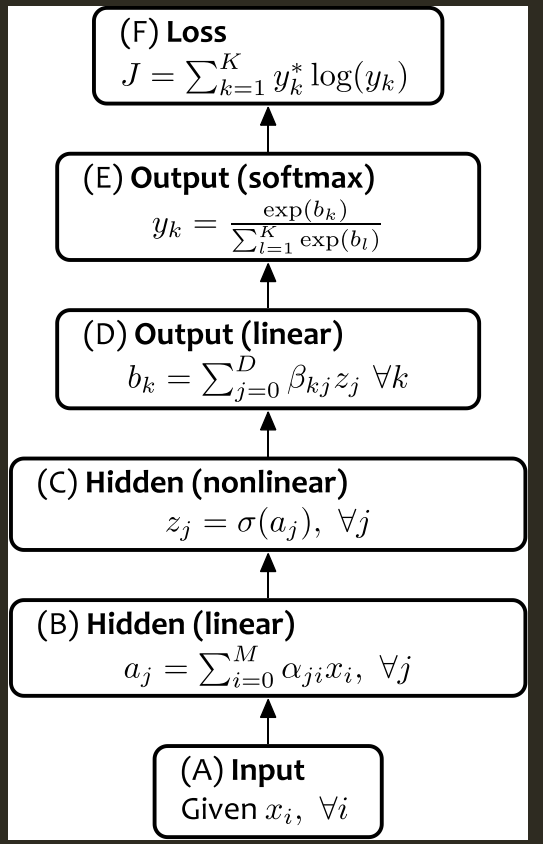
MULTI-CLASS OUTPUT



MULTI-CLASS OUTPUT

Softmax: $$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$





KERAS AND ANN (2)

Models created by Keras can be executed on a **backend**

- **Tensorflow (default)**
- **Theano**

Keras has built in GPU support with CUDA

Keras is easier than Tensorflow

KERAS COMPONENTS

Most important component in KERAS is **models**

Models = layers + loss + optimizer

After adding these, you call compile() and fit()

Models can be saved and checkpointed

KERAS LAYERS

Layers are used to define the architecture

- Dense layers (this is the normal, fully-connected layer)
- Dropout layers (these are used for regularization, to avoid overfitting)
- Convolutional layers (applies convolution operations on the previous layer)
- Pooling layers (used after convolutional layers)
- Embedding layer
- LSTM layer

KERAS LOSS FUNCTIONS

Loss functions are used to compare the network's predicted output with the real output, in each pass of the backpropagations algorithm

Loss functions are used to tell the model how the weights should be updated

- Mean squared error
- Cross entropy
- etc.

FEW POINTS WHILE DESIGNING THE NETWORK (1)

1. Momentum to tackle local minima: Momentum is a term that incorporates the properties from the previous weight update to allow the weights to continue to change in the same direction even when there is less error being calculated.
2. Learning rate decay : learn more in beginning and slowly as you tend to converge
3. Stratified vs K fold : We are NOT using K fold cross validation becuase it is very costly. Instead we are using StratifiedKfold from scikitlearn. The folds are stratified, meaning that the algorithm attempts to balance the number of instances of each class in each fold. It creates and evaluates 10 models using the 10 splits of the data and collects all of the scores. ScikitLearn cross_val_score can take either KFold or StratifiedKFold as parameters.

FEW POINTS WHILE DESIGNING THE NETWORK (2)

4. Are we standardizing data : It is almost always good practice to standardize data before modeling it using a neural network model. We set mean 0 and std deviation 1.
5. Use scikit pipeline : Pipeline is a scikitlearn wrapper. Within this, we standardize the data and do execute the ML model in one pass of the cross-validation.
6. Verbose output: Verbose output is set by verbose=1. This returns the accuracy (model is previously configured as metric=accuracy). This array is used to print out mean and standard deviation.

FEW POINTS WHILE DESIGNING THE NETWORK (3)

7. Deeper or broader network: You can try this out : Drop out as regularization technique : The paper on dropout gives certain heuristics such as (a) drop out has good effect on large network (b) can do this ion both visible and hidden layers © do it with large training rate with decay (while constraining the weights)and large momentum
8. No of training epochs : in the early stages of experimentation it can be helpful to turn off early stopping, so you can see any signs of overfitting, and use it to inform your approach to regularization. A good rule for early stopping is to terminate if the best classification accuracy doesn't improve for quite some time. (for some epochs, say 10)

FEW POINTS WHILE DESIGNING THE NETWORK (4)

9. Try out the optimizers : Optimizers are strategies used to update the network's weights in the backpropagation algorithm. The most simple optimizer is the Stochastic Gradient Descent Algorithm (SGD), but there are many other (RMSProp, Adagrad) . The other important thing is how you initialize the weights. Most optimizers can be tuned using hyper parameters, such as (Learning rate, momentum etc.)
10. Do you have GPU / CUDA set up ? If your computer has a good graphics card, it can be used to speed up model training

SAMPLE KERAS CODE FOR CREATING ANN

```
def create_ANN_model():
    # create model
    model = Sequential()  # model class provided by Keras
   # 60 input independent variables , connected to 60  neurons in next  layer
    model.add(Dense(60, input_dim=60, init='normal', activation='relu')) model.add(Dense(60, init='normal', activation='relu'))
   #hidden layer #1 with RELU activation, densely connected with input neurons, connected to 60  neurons in next  layer
    model.add(Dense(30, init='normal', activation='relu'))
   #hidden layer #2 with RELU activation, densely connected with input neurons, connected to 30  neurons in next  layer which is an output layer
    model.add(Dense(1, init='normal', activation='softmax')) #output layer with 1 output and  sigmoid activation function
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])  # loss function is cross entropy ( binary), optimizer is SGD etc.
    return model
```

CLASSIFIER EVALUATION METRICS: ACCURACY, ERROR RATE, SENSITIVITY AND SPECIFICITY

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Classifier Accuracy, or recognition rate: percentage of test set tuples that are correctly classified

Accuracy = (TP + TN)/All

Error rate: *1 – accuracy*, or

Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

CLASSIFIER EVALUATION METRICS: PRECISION AND RECALL, AND F-MEASURES

Precision: exactness – what % of tuples that the classifier labeled as positive are actually positive

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

Recall: completeness – what % of positive tuples did the classifier label as positive?

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

Perfect score is 1.0

Inverse relationship between precision & recall

***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

***F_β*:** weighted measure of precision and recall

- assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$