Course

The representation of a course, containing `id`, `name`, `questions`, `answers`, `transcripts` and `text` as properties.

Constructor

The constructor takes each of its properties in order as parameters, all having default values of `null` apart from `id`.

```
$course = new Course("906", "5G System Engineering");
// creates course with given id and name, other properties are null.
```

Set Property

To set a property, for example `name`, use `setName`.

```
$course = new Course("906");
$course->setName("5G System Engineering");
```

Get Property

To get a property, for example `name`, use `getName`.

```
$course = new Course("906");
$course->getId();
//returns "906"
```

Get Data

To get all data from a course, use `getData`. The merging of `questions`, `answers`, `transcripts` and `text` is returned as an array.

```
$course = new Course("101","example",["question1"],["answer1"],["trascript1"],["text1","text2"]);
$course->getData();
//returns ["question1","answer1","transcript1","text1","text2"]
```

Is Complete

To determine whether a course has no null properties, use `isComplete`. Returns true if course has no null properties, false if course has atleast 1 null property.

```
$course = new Course("111");
$course->isComplete();
//return false since only `id` is set
```

Course Retriever

An object used to retrieve course data from an sql database. The database is accessed through php data objects.

Get Courses

To retrieve all courses, use `getCourses`. Parameters include `identifiersOnly` and `material`, the former used to decide whether only identfying data will be retrieved (e.g. `id` and `name`) and the latter to decide what data is retrieved.

```
$cr = new CourseRetriever;
$cr->getCourses();
// returns array of courses
```

Get Name

To get the `name` of a course using its `id`, use `getName`.

```
$cr = new CourseRetriever;
$cr->getName("906");
// returns "5G System Engineering"
```

Data

An `interface` for data that should be stored in memory. The interface is comprised of 3 methods: `getData`, `getPath` and `isValid`.

Network Data

An implementation of Data, contains `path`, `inputSize`, `layers`, `iterations`, `activation` and `targetSet`. Used to store data about the neural network.

```
$networkData = new NetworkData;
$networkData->path;
// returns "network//meta.txt"
```

Storage Data

An implementation of Data, contains `path`, `wordCount`, `courses`, `material` and `trainingPercentage`. Used to store data about the samples.

```
$storageData = new StorageData;
$storageData->path;
// returns "samples//meta.txt"
```

Get Data

To get raw data of an object implementing the `Data interface`, use `getData`. The result is conventionally the data contained in the objects properties merged into a string seporated by new-lines for the purpose of writing to memory.

```
$storageData = new StorageData;
$storageData->wordCount = 10;
$storageData->course = ["906","968","1026"];
$storageData->material = ["pdfs"];
$storageData->trainingPercentage = 20;
$storageData->getData();
// returns "10\n906,968,1026\npdfs\n20"
```

Get Path

To get the path to storage location of an object implementing the `Data interface`, use `getPath`.

```
$storageData = new StorageData;
$storageData->getPath();
// returns "samples//meta.txt"
```

Is Valid

To determine whether an object implementing the `Data interface` has valid properties, use `isValid`.

```
$storageData = new StorageData;
$storageData->isValid();
// returns 'false' due to null properties
```

Network Manager

Used to create, retrieve update or delete stored networks.

Get Network

Store Network

Get Storage

Predict

Set Active Network

Get Active Network

Get Network Data

AI Search

A state machine used to load data from the database and to train, test and store a network.

The network is represented as a pipeline containing TokenCountVectorizer, TF-IDFTransformer and a NaiveBayes classifier.

The network is stored in memory using the ModelManager.

This is the only class that uses with php-ml.

Get State

To get the state of AI Search, use `getState`. The possible states are `"nothing"`, `"loaded"`, `"vectorized"` and `"trained"`. The current state will be loaded from memory during construction.

```
$ai = new AISearch;
if($ai->getState() == "nothing")
{
    echo "ready to read.";
}
else if($ai->getState() == "trained")
{
    echo "ready to test.";
}
```

Set Data

Meta data for the storage and network can be set through `setStorageData` and `setNetworkData` respectively. An object implementing `Data` must be passed and is set if valid.

```
$ai = new AISearch;
$ai->setStorageData($storageData); //works if($storageData->isValid())
$ai->setNetworkData($networkData); //works if($networkData->isValid())
```

Train and Test Network

To retrive, train on data and test the network, use `trainAndTestNetwork` with parameters `NetworkData` and `StorageData` the default values of which are `null`.

The returned `float` is the accuracy of the network, `0` being completely inaccurate and `1` being totally accurate.

```
$ai = newAISearch;
$ai->trainAndTestNetwork();
//returns accuracy as float 0-1
```

Predict

To predict what course a query will return, use the `predict` function. This function loads the network from memory (if not already loaded) and returns the prediction for the given input. To load the network ahead of time, preceed `predict` with `primePredictor`.

```
$ai = new AISearch;
$ai->primePredictor(); //not necassary but reduces time for predict to execute.
$ai->predict($query);
//returns course ID of prediction for $query
```

Read Data

To read and store data from the database, use `readData`. If the state of AI Search is `"nothing"`, course data is read from a database using `CourseRetriver` and stored under `data/*courseID*/`. Finally, state is set to `"loaded"`.

```
$ai = new AISearch;
if($ai->getState() == "nothing")
{
    $ai->readData();
    //$ai->getState() returns "loaded"
}
```

DoTFIDF

To do the tf-idf seperately from training, use `doTFIDF`. If the state of AI Search is `"loaded"`, the data will be vectorized and stored in `samples/raw`.

```
$ai = new AISearch;
if($ai->getState() == "loaded") $ai->doTFIDF();
```

Train Network

Vectorizing and Training

To train the network on stored data, use `trainNetwork`. If the state of AI Search is `"loaded"`, data is split into training and testing sets based on the `storedData->trainingPercentage`. Training set is used to train, testing set is stored for testing at `samples/`. Finally, state is set to `"trained"`.

Training involves Tokenization of data with NGramTokenizer and english stop words, before being transformed into TF-IDF vectors and used to train a NaiveBayes network.

```
$ai = new AISearch;
if($ai->getState == "loaded")
{
    $ai->trainNetwork();
    //$ai->getState() returns "trained"
}
```

Pre-Vectorized Training

Alternatively, if the state of AI Search is `"vectorized"`, `trainNetwork` will retrieve the stored samples and train with those. The network stored still contains the necassary transformers for prediction from string.

```
$ai = new AISearch;
if($ai->getState() == "vectorized")
{
    $ai->trainNetwork();//trains with stored, vectorized samples
}
else if($ai->getState() == "loaded")
{
    $ai->trainNetwork();//uses php-ml/pipeline to vectorize and train in one
}
```

Test Network

To test the network on stored testing samples, use `testNetwork`. If the state of AI Search is `"trained"`, the method will read testing samples from file and test the network, returning accuracy as a `float` in `(0,1)`.

```
$ai = new AISearch;
if($ai->getState() == "trained")
{
    echo "Accuracy: " . $ai->testNetwork() . PHP_EOL;
}
```