En el documento Preprocessing se define una acción la cual comentaremos a continuación:

En primer lugar, se nombra la clase *File*. Esta clase básicamente nos proporciona información acerca de los archivos, de sus atributos, de los directorios, etc. La clase *File* tiene tres constructores

* *File(String path)*
* *File(String path, String name)*
* *File(File dir, String name)*

El parámetro *path* indica el camino hacia el directorio donde se encuentra el archivo, y *name* indica el nombre del archivo. En este caso se abre el archivo "r" que se encuentra en el directorio "static/default of credit card clients.csv"

Inmediatamente después se utiliza el objeto *out* para sacar por pantalla el archivo "w" del directorio "static/dataset.csv".

Después, para leer el archivo línea por línea a String, se utiliza Readline. Este método devuelve nulo cuando se llega al final del archivo. En este documento se le ha denominado *firstline.*

La clase *String* ofrece un método *split ()* que es muy conveniente y suficiente para la mayoría de los escenarios. Simplemente divide la *cadena* dada en función del delimitador, devolviendo una matriz de *cadenas*. Por otro lado, strip() elimina los espacios en blanco del principio y del final de la cadena.

Creamos una cadena: (string=""). Las cuerdas son constantes; sus valores no se pueden cambiar después de que se crean. Los búferes de cadena admiten cadenas mutables. Debido a que los objetos String son inmutables, se pueden compartir.

A continuación, utilizamos los bucles para meter los valores en la cadena anterior. La declaración de for proporciona una forma compacta de iterar sobre un rango de valores. Los programadores a menudo se refieren a él como el "bucle for" debido a la forma en que se repite repetidamente hasta que se cumple una condición particular.

Un primer bucle, de rango 0 a 6, se va introduciendo los valores, junto con un punto y coma, en la cadena, por medio del array args. De la misma forma se hace con los array de 12 a 24. Finalmente, una vez introducido el número 24 en el array, se muestra por pantalla la cadena dividido por n.

En el siguiente for se vuelve a utilizar los métodos Split (para dividir la cadena) y Strip para eliminar los espacios en blanco. Únicamente se introduce los valores en la cadena de 25 elementos si es verdadero la condición que se ha establecido. Es decir, siempre que el rango sea de 12 a 24 y además sea de valor 0, no se cumplirá esta condición y no se insertan los valores en el array.

Finalmente, se cierra el fichero y se devuelve un 0.

Esta acción se utilizará cuando el nombre sea “\_\_main\_\_”

En el último documento realizamos lo siguiente: **server.py**

En primer lugar, para crear rutas web de una forma muy sencilla utilizamos Flask:

**“from** flask **import** Flask” a la que posteriormente se le debe citar: app = Flask(\_\_name\_\_, static\_folder='/static');

Para importar las clases, ya definidas en los archivos anteriores: “Flask, render\_template, send\_from\_directory, request y jsonify”, utilizamos la palabra clave “import”.

También se importa las clases “numpy” y “pandas”, como “np” y “pd” respectivamente.

Posteriormente, importamos las clases “svm” de “sklearn”, “cross\_val\_score” de “sklearn.model\_selection” y frequency, instances, pca Standardizatio y preprocesing. En esta última ocasión, como se puede apreciar, no hemos declarado paquete alguno para nuestra clase ya que únicamente hemos realizado el import, sin embargo, este import como tal es algo particular para nosotros, pues hemos importado TODAS las clases pertenecientes al paquete.

Para dejar en blanco “clf” lo igualamos a “none”. “last\_id” lo igualamos a la cifra 30000.

A continuación, se definen las apliaciones web:

|  |
| --- |
| @app.route("/") |
|  | def index(): |
|  | return render\_template("charts.html") |

En primer lugar, debemos tener en cuenta que Chart.html es una biblioteca de javascript para crear gráficos simples y limpios, Todos ellos están basados ​​en HTML5, responsivos, modulares, interactivos y hay un total de 6 gráficos.

También se ha de tener en cuenta que, en los archivos anteriores se ha creado la aplicación web, la cual cuando se le llama muestra el gráfico correspondiente y se denomina index. We simply pass the arrays to render\_template(). This means that most of the magic occurs in the template. Chart.js is a client-side javascript library which is why our app.py is very minimal.

Algo parecido realizamos para enviar un archivo desde un directorio dado con [send\_file()](https://kite.com/python/docs/flask.send_file). Esta es una forma segura de exponer rápidamente archivos estáticos desde una carpeta de carga o algo similar. En nuestro proyecto:

|  |
| --- |
| @app.route('/<path:path>') |
|  | def send\_js(path): |
|  | return send\_from\_directory('static', path) |

|  |  |
| --- | --- |
| Parameters: | * **directory** – the directory where all the files are stored: static * **filename** – the filename relative to that directory to download: path * **options** – optional keyword arguments that are directly forwarded to [send\_file()](https://kite.com/python/docs/flask.send_file). |

La última aplicación web sirve para obtener los diferentes valores, es ese el motivo por el que se le ha dicho nombre: “getValues”

El comando “Request.Form” se usa para recopilar valores en un formulario con “method” = "post”” (cómo interactuar con el usuario).

La información enviada desde un formulario con el método POST es invisible para otros y no tiene límites en la cantidad de información que se debe enviar. En este caso los formularios son “key” y “value”.

Posteriormente, se define la variable global “last\_id” y, llamando a las diferentes rutas anteriores, se les da valores a las variables. Finaliza devolviendo en pantalla la cadena mediante la declaración de “return”.

Finalmente:

*“argv”*represents all the items that come along via the command-line input, but counting starts at zero (0) not one (1), len(sys.argv) ¡= 2 just checks que no sea igual que dos.

Muesta por pantalla: You have to specify if you want the accuracy or not! Use 0 if not, 1 if yes.

Exit (-1) generalmente se utiliza para indicar una terminación incorrecta.

Una vez introducido dicha condición, comienza la fase de aprendizaje llamando a las correspondientes acciones ya definidas anteriormente.

Las características de los datos son "SEX", "EDUCATION", "MARRIAGE", "AGE", "BILL\_AMT1", "BILL\_AMT2", "BILL\_AMT3", "BILL\_AMT4", "BILL\_AMT5", "BILL\_AMT6", "PAY\_AMT1", "PAY\_AMT2", "PAY\_AMT3", "PAY\_AMT4", "PAY\_AMT5", "PAY\_AMT6"

Read\_csv: is an important pandas function to read csv files and do operations on it.

Opening a CSV file through this is easy. But there are many others thing one can do through this function only to change the returned object completely. En este caso el delimitador es “;”.The header is where you tell Java what value type (0, in this case), if any, the method will return (an int value, a double value, a string value, etc).

En x introducimos los datos numéricos de la base de datos donde se encuentran las características pero en formato de matriz con np.array().

En y introducimos los valores de los datos “default payment nex month” en formato de lista.

En cuanto a clf = svm.SVC(kernel="rbf", verbose=True) the fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples.

**kernel : *string, optional (default=’rbf’)***

Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used.

**verbose : *bool, default: False***

Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

The clf (for classifier) estimator instance is first fitted to the model; that is, it must *learn* from the model. This is done by passing our training set to the fit method

EN cuanto a sys.argv, is automatically a list of strings representing the arguments (as separated by spaces) on the command-line. The name comes from the [C programming convention](http://www.crasseux.com/books/ctutorial/argc-and-argv.html) in which argv and argc represent the command line arguments.

To use it, you will first have to import it (import sys) The first argument, sys.argv[0], is always the name of the program as it was invoked, and sys.argv[1] is the first argument you pass to the program.

Cuando se cumple que “sys.argv[1]=="1"”:

1. Se imprime por pantalla el siguiente mensaje: "Evaluating performances"
2. Mediante “cross\_val\_score” we evaluate a score by cross-validation. The parameters are:
   1. **estimator : *estimator object implementing ‘fit’ (CLF)***

The object to use to fit the data.

* 1. **X : *array-like***

The data to fit. Can be for example a list, or an array.

* 1. **y : *array-like, optional, default: None***

The target variable to try to predict in the case of supervised learning.

* 1. **scoring : *string, callable or None, optional, default: None***

A string (see model evaluation documentation) or a scorer callable object / function with signature scorer(estimator, X, y) which should return only a single value.

Similar to **[cross\_validate](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html" \l "sklearn.model_selection.cross_validate" \o "sklearn.model_selection.cross_validate)** but only a single metric is permitted.

If None, the estimator’s default scorer (if available) is used.