Article
Henry Pereira · Apr 6, 2022 7m read

So, Where's my money?

All of us know that money is important. We constantly need to monitor all expenses to avoid looking back to the bank statement and thinking: “ So, where ’ s my money? ”

To evade financial stress, we must keep an eye on the inflow and outflow of money into our accounts.It is also important to tack when and how we spend and earn. Manually recording all transactions in order to understand where our money goes requires an effort. It demands consistency, and it is boring. Today there is a bunch of mobile or SaaS options that help you manage your finances.

I believe that there ’ s a good chance you are doing this already.
Yet, how about developing an expense tracker application that will keep an eye on our expenses whereas we will just need to feed it with the information?

In this article, we will build an expense tracker app using embedded Python with Scikit-learn for machine learning to help you categorise all transactions.

We're going to call this project "soWhereIsMyMoney". If you started reading this article imagining that I would complain about something, sorry if that sounds like clickbait.

Buckle up everyone, and let's go!

For this project, I used the InterSystems-iris-dev-template, it is a batteries-included configuration for docker and IRIS.

The next step is to add the python libraries that we need to Dockerfile.

```
ENV PIP_TARGET=${ISC_PACKAGE_INSTALLDIR}/mgr/python

RUN pip3 install numpy \
        pandas \
        scikit-learn
```

The flow of money between accounts is represented by transactions grouped by categories.

Now, we are going to create two persistent classes with a one-to-many relationship: Category and Transaction.

```
Class dc.soWhereIsMyMoney.Category Extends %Persistent
{
Index IdxName On Name [ Unique ];
Property Name As %String [ Required ];
Property Type As %String(DISPLAYLIST = ",Income,Expense", VALUELIST = ",I,E") [ Initi
alExpression = "E", Required ];

Relationship Transactions As dc.soWhereIsMyMoney.Transaction [ Cardinality = many, In
verse = Category ];
}
```

Categories are composed of two types, where one is the Income and the other one is the Expense.

```
Class dc.soWhereIsMyMoney.Transaction Extends %Persistent
{
Index IdxCategory On Category;
Property Description As %String [ Required ];
Property CreatedAt As %DateTime;
Property UpdatedAt As %DateTime;
Property Amount As %Numeric [ Required ];

Relationship Category As dc.soWhereIsMyMoney.Category [ Cardinality = one, Inverse =
Transactions ];
}
```

The category will help you keep in touch with spending. There are a million categories to consider. Everything should be accounted for, from large expenses like your mortgage and car payment to smaller ones like your Netflix subscription.
Some users might choose to group “ Lunch ” , “ Eating out ” and “ Takeaway ” into one single “ Food ” category. Other ones might split them between “ Eating out ” and “ Groceries. ”

It ’ s time to define the categories into which all the transactions should fall. To keep this sample simple, I chose to use only 8 categories:

- Income
- Eating Out

- Personal Care
- Fuel
- Sport
- Utilities
- Entertainment
- Groceries

Our goal is to give you a description. We want to assign it to one of these categories. It can be done using the power of Machine Learning.

The extensive prevalence of text classification articles and tutorials on the Internet is related to binary text classification like sentiment analysis.
The challenge is that we have to deal with a multiclass text classification problem.
To solve that, we can employ some different machine learning models like Multinominal Naive Bayes, Random Forest, Logistic Regression, Linear Support Vector Machine, etc.
For our case, Linear Support Vector Machine will fit perfectly.

If you want a simple explanation about Support Vector Machine, I strongly recommend watching SVM in 2 minutes video.

Ok, it has been a bit too fast till now. It certainly would be good to step back for a moment and give you a more detailed explanation of the text representation.
Classifiers can not directly process the text documents in their original form. First, it is necessary to preprocess the text converting it to a more numerical representation.

As I said before, we will use Scikit-learn. Scikit-learn is a Python library that contains such tools as regression, clustering, dimensionality reductions, and classification for statistical modelling and machine learning.

So, will transfer all records from "Transaction" into a pandas Dataframe.

```
import iris
rs = iris.sql.exec('SELECT Category->ID as category_id, Category->Name as category, D
escription as description FROM dc_soWhereIsMyMoney."Transaction" ')
df = rs.dataframe()
```

Noice, it's very simple to do.

```
from sklearn.model_selection import train_test_split
  X_train, _, y_train, _ = train_test_split(df['description'], df['category'], random
_state = 0)
```

We have just create a train-test split (part of the data set is left out from the training to see how the model performs on data it hasn ’ t seen).
The **xtrain** defines the part of the data that help us make the predictions. It is the input (the transaction descriptions). The **ytrain** defines the expected categories for those inputs.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(min_df=5, encoding='utf8', ngram_range=(1, 2), stop_word
s='english')
  X_train_counts = count_vect.fit_transform(X_train)
```

CountVectorizer will transform a collection of descriptions into a matrix of token counts.

mindf is the minimum frequency rate a word must have in the input data in order to be kept, **ngramrange** is set to (1, 2) to indicate that we want to consider both unigrams and bigrams.
To reduce the number of noisy features **stopwords** is set to "English" to remove all common pronouns and articles ("a", "the", ...).

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer(norm='l2',sublinear_tf=True)
  X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

Then we will transform these word counts into frequencies. For each term in our dataset, we will calculate a measure called "Term Frequency, Inverse Document Frequency", abbreviated to tf-idf.
To ensure all our feature vectors have a Euclidian norm of 1, the **norm** is set to l2 and **sublineardf** is set to True to be able to use a logarithmic form for frequency.

After all data transformations we have made, now we have all the descriptions and categories. Thus, it is time to fit the model to train the classifiers.

```
from sklearn.svm import LinearSVC
model = LinearSVC()
model.fit(X_train_tfidf, y_train)
```

In the source code, which can be found on Github, I left a fake dataset that I used to train and test the project.

This is a full description of the method to suggest a category:

```
Class dc.soWhereIsMyMoney.nlp.Category
{

ClassMethod Suggestion(description As %String) [ Language = python ]
{
  import iris
  from sklearn.model_selection import train_test_split
  from sklearn.feature_extraction.text import CountVectorizer
  from sklearn.feature_extraction.text import TfidfTransformer
  from sklearn.svm import LinearSVC

  X_train, _, y_train, _ = train_test_split(df['description'], df['category'], random
_state = 0)
  count_vect = CountVectorizer(min_df=5, encoding='utf8', ngram_range=(1, 2), stop_wo
rds='english')
  X_train_counts = count_vect.fit_transform(X_train)
  tfidf_transformer = TfidfTransformer(norm='l2',sublinear_tf=True)
  X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
  model = LinearSVC()
  model.fit(X_train_tfidf, y_train)

  return model.predict(count_vect.transform([description]))[0]
}

}
```

For the Transaction class, we need to add a method of allocating all input data into the right category.

```
ClassMethod Log(amount As %Numeric, description As %String, category As %String = "")
 As %Status
    {
     Set tSC = $$$OK
     Set transaction = ..%New()
     Set transaction.Amount = amount
     Set transaction.Description = description
     Set:(category="") category = ##class(dc.soWhereIsMyMoney.nlp.Category).Suggestio
n(description)
     Try {
     $$$ThrowOnError(##class(dc.soWhereIsMyMoney.Category).FindOrCreateByName(categor
y, .Category))
     Set transaction.Category = Category
     $$$ThrowOnError(transaction.%Save())
     } Catch ex {
     Set tSC=ex.AsStatus()
     }
     Quit tSC
    }
```

That was it! Next time we can dive deeper and get acquainted with all possibilities the power of ePython could offer us.

Please do reach out to me with all the suggestions, comments, and feedback you may have.

Github link to this code

Thanks for reading.

#Artificial Intelligence (AI) #Embedded Python #Vector Search #InterSystems IRIS