# Tuto5: Developing a 2D-projection of documents in a scatterplot

Nicolas Médoc Luxembourg Inst. of Science and Technology

# Outline

1. Text processing with Python Flask server
2. Calling Python Flask server API from React
3. Coloring per categories
4. Comparing different projections

# Setting up a Flask server for text processing

- Install Python 3.12: `https://www.python.org/downloads/`
- Install PyCharm Community Edition or any other Python IDE: `https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows&code=PCC`
- Getting git repository with predefined code for text processing `https://github.com/nicolasmedoc/TD5-text-Python`

# Setting up a Flask server for text processing

- Create virtual environment when opening the project in PyCharm
- or create it manually
  - python -m venv /path/to/new/virtual/environment
- Installing dependencies with PyCharm:
  - open the menu Tools->Sync Python requirements
  - Click on install requirements
- or with pip in a terminal:
  -

# Setting up a Flask server for text processing

Create server.py:

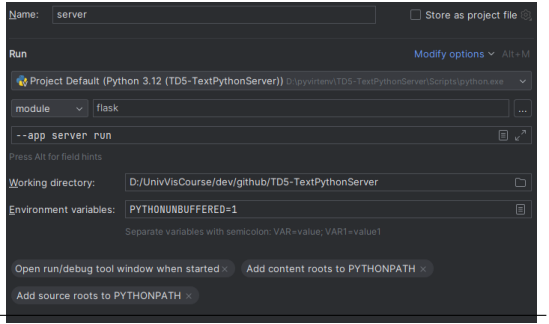
```
from flask import Flask,request
from flask_cors import CORS

app = Flask(__name__)
# allows cross origin to be called from localhost:3000
# not recommended in production
CORS(app)

# insert code for server initialization if needed

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

# Setting up a Flask server for text processing

- In a terminal:

```
python -m flask --app server run
```

- In PyCharm:
  right click on server.py...
- Test:
  `http://localhost:5000/`
- Shows "Hello, World!"



# Text processing at server initialization

```
import dataset
import textprocessing
import dimred
import projection
...
# insert code for server initialization if needed
dataset, true_k = dataset.get20newsgroups()
x_tfidf, vectorizer = textprocessing.get_tfidf(dataset.data)
x_lsa, lsa = dimred.lsa(x_tfidf)
proj_euclidean = projection.tsne_euclidean(x_lsa)
...
@app.route("/getProjection")
def get_projection():
    return {'projection': proj_euclidean.tolist(), 'categories': dataset.target.tolist()}
```

Test: `http://localhost:5000/getProjection`

Shows: "categories":[0,1,1,...],"projection":[[30.4295158386230 47,24.48200798034668],...]

# React: using createAsyncThunk

https://github.com/nicolasmedoc/TD5-text-React
In redux/DataSetSlice: add asynchronous reducers with createAsyncThunk (e.g. to call external API)

```
export const getProjectionData = createAsyncThunk('projectionData/fetchData',
    async (args,thunkAPI) => {
        const response = await fetch('http://localhost:5000/getProjection');
        const responseJson = await response.json();
        return responseJson.projection.map((item,i)=>{
            return { xValue:item[0], yValue:item[1],
                     index:i, category:responseJson.categories[i]}
        });
        // when a result is returned, extraReducer below is triggered
        // with the case getProjectionData.fulfilled
})
```

# React: using createAsyncThunk

Declare the extraReducers to control the lifecycle of the async request when pending/fullfilled/rejected

```
export const dataSetSlice = createSlice({
  name: 'dataSet',
  initialState: [],
  reducers: {       // add reducer if needed
  },
  extraReducers: builder => {
    builder
        .addCase(getProjectionData.pending, (state, action)=>{})
        .addCase(getProjectionData.fulfilled, (state, action) => {
          return action.payload
        })
        .addCase(getProjectionData.rejected, (state, action)=>{})
  }
})
```

# React: dispatch async reducer from a React component

## Add a "did mount" useEffect in App.js

```
import { getProjectionData } from './redux/DataSetSlice';
import { useDispatch } from 'react-redux';
...
function App() {
    const dispatch = useDispatch();
    ...
    useEffect(()=>{
      dispatch(getProjectionData());
    },[]) // empty dependencies [] <=> component did mount
    ...
}
```

## You will see the scatterplot populated

# React: adding color in scatterplot-d3

In create() function

```
this.colorScale = d3.scaleOrdinal(d3.schemeObservable10)
```

In updateAxis() function

```
const categories = Array.from(new Set(visData.map(item=>item.category))
categories.sort()
this.colorScale.domain(categories);
```

In updateDots() function

```
selection.select(".dotCircle")
    .attr("fill",(item)=>{
        return this.colorScale(item.category);
    })
```

# Python Server: build different projections

In Server.py, add two other projections:

```
proj_cosine = projection.tsne_cosine(x_tfidf)
proj_euclidean_tfidf = projection.tsne_euclidean_tfidf(x_tfidf)
```

return the right projection depending on 'distance' parameter:

```
@app.route("/getProjection")
def get_projection():
    distance = request.args.get('distance');
    proj = None
    if distance=='euclidean':
        proj = proj_euclidean
    elif distance=='cosine':
        proj = proj_cosine
    elif distance == 'euclidean_tfidf':
        proj = proj_euclidean_tfidf
    return {'projection': proj.tolist(), 'categories': dataset.target.tolist()}
```

# React: add ’distance’ parameter when calling Python API

In redux/DataSetSlice.js:

```
const params = new URLSearchParams(args).toString();
const response = await fetch('http://localhost:5000/getProjection?'+params);
```

In App.js, remove the dispatch call in useEffect()
Add the ControlBar component to select the projection to display

```
import ControlBar from "./components/ControlBar/ControlBar";
...
  return (
    <div className="App">
      {console.log("App rendering")}
      <div id={"control-bar-container"}>
        <ControlBar/>
      </div>
...
```