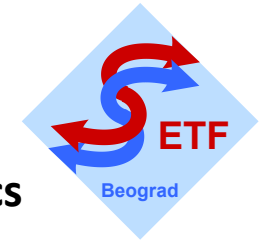
University of Belgrade
School of Electrical Engineering
Subject: **Computational Genomics**



Detection of spatially variable genes

Students:
Aleksandar Cvetković, 2022/3270
Kristina Stanković, 2022/3019

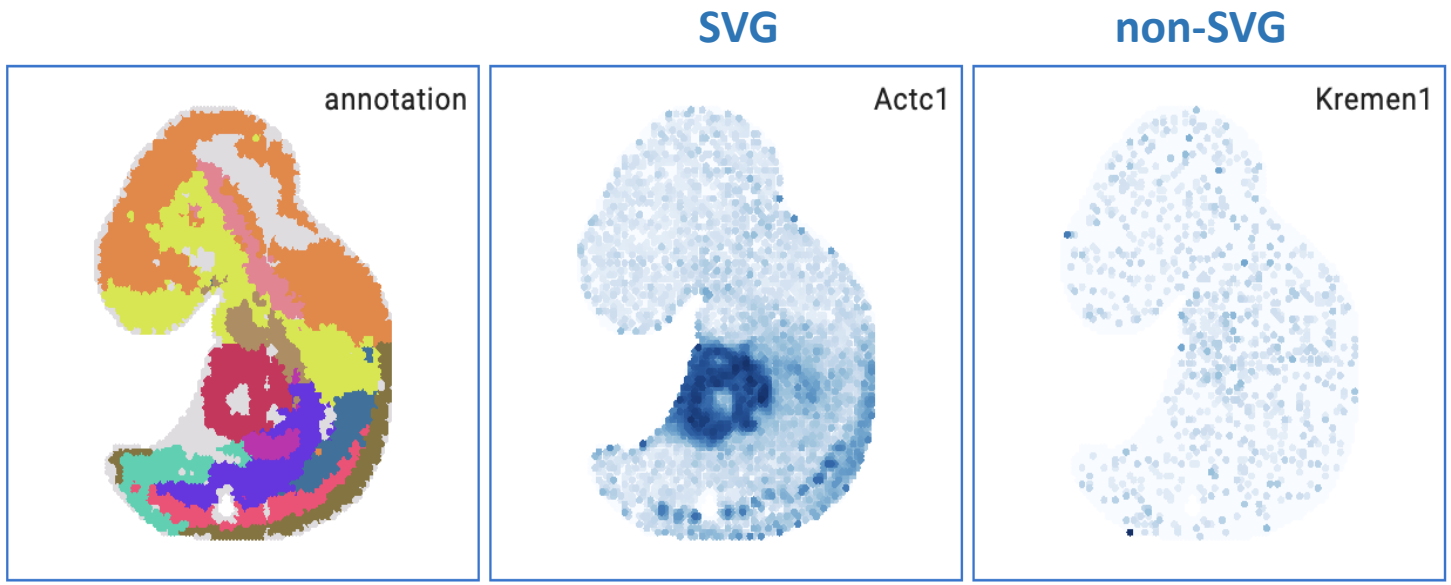
Overview of the project

- Purpose: Identifying spatially variable genes (SVGs)
- Methods:
    1. Algorithm: Combination of Mean difference values and Spatial clustering
    2. Algorithm: Combination of Variance of entropy values and Spatial clustering
    3. Algorithm: Combination of Variance of mean values and Spatial clustering
    4. Algorithm: Moran's I
    5. Graph Fourier transform framework (SpaGFT)
- Testing the algorithm on Mouse embryo 9.5 and Mouse brain samples

Spatially variable genes (SVGs)

- Difference between highly variable genes (HVGs) and spatially variable genes (SVGs)

SVGs are defined as genes with a highly spatially correlated pattern of expression, which varies along with the spatial distribution of a tissue structure of interest
HVGs are defined purely based on molecular features (i.e. gene expression), and do not take any spatial information into account

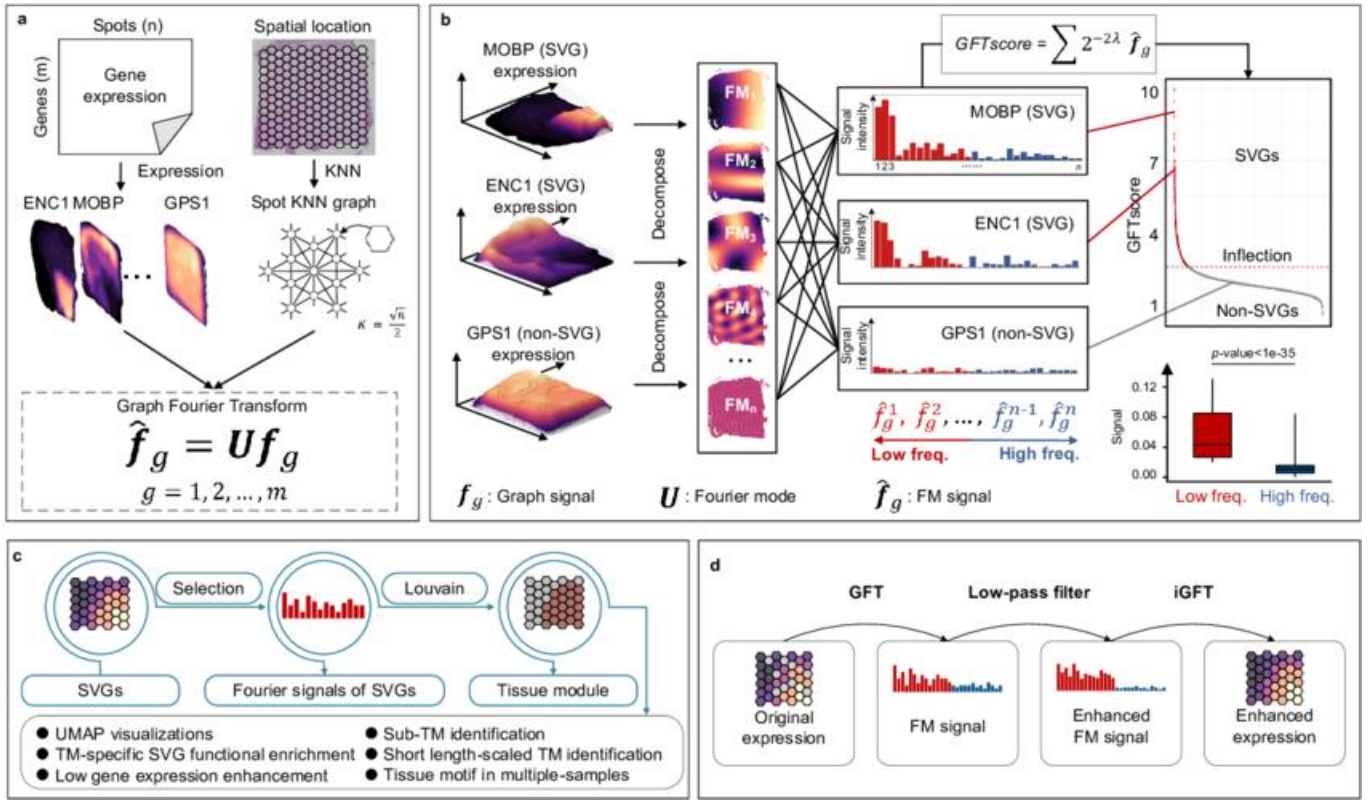


Mouse embryo 9.5

SpaGFT

Python package to analyze tissue functions empowered using spatial omics data

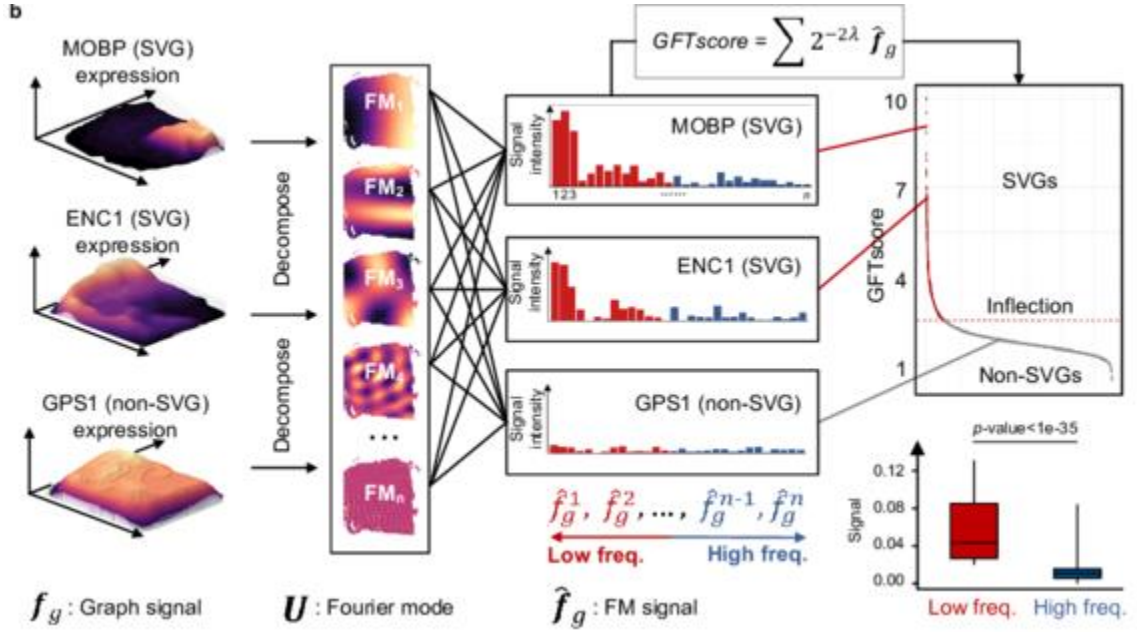
SpaGFT is a hypothesis-free graph Fourier transform framework (GFT) for SVG identification from spatial transcriptomics data without assuming any spatial distribution patterns.



SpaGFT

SpaGFT (2)

Rule: A gene with a high intensity of low-frequency FM signals compared to high-frequency FM signals is typically an SVG, whereas a gene with a low intensity of low-frequency FM signals indicates random expression patterns.



SpaGFT

SpaGFT (3)

Implementation:

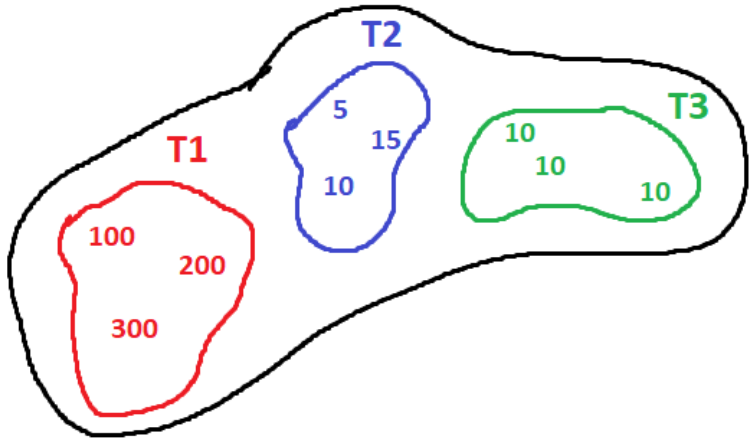
```
# determine the number of low-frequency FMs and high-frequency FMs
(ratio_low, ratio_high) = spg.gft.determine_frequency_ratio(adata, ratio_neighbors=1, spatial_info='spatial')
# calculation
gene_df = spg.detect_svg(adata,
                         spatial_info='spatial',
                         ratio_low_freq=ratio_low,
                         ratio_high_freq=ratio_high,
                         ratio_neighbors=1,
                         filter_peaks=True,
                         S=6)
# S determines the  sensitivity of kneedle algorithm
# extract spaitally variable genes
svg_list = gene_df[gene_df.cutoff_gft_score][gene_df.qvalue < 0.05].index.tolist()
print("The number of SVGs: ", len(svg_list))
```

Tutorial

Algorithm: Mean difference value

The idea of this algorithm is that there should be different mean values on tissues which contain SVG and tissues that do not contain it.

For example, the mean value in tissue T1 is 200, while the mean value outside of T1 (in tissues T2 and T3 combined) is 10, and the difference between them is 190, while the mean value in tissue T2 is 10 and outside of the T2 tissue (in tissues T1 and T3) the mean value is 105 and difference between them is 95. Difference should be higher for tissues which contain SVG. In order to use the same threshold for all genes, gene expression should be **normalized** (this is not done is this simple example).



Mean in T1 is: (100 + 200 + 300) / 3 = 200
Mean outside of T1 is: (5 + 10 + 15 + 10 + 10 + 10) / 6 = 10

Mean in T2 is: (5 + 10 + 15) / 3 = 10
Mean outside of T2 is: (100 + 200 + 300 + 10 + 10 + 10) / 6 = 105

Mean in whole organism is: (100 + 200 + 300 + 5 + 10 + 15 + 10 + 10 + 10) / 9 = 73.3

For tissue T1 mean difference value is (200 – 10) / 73.3 = 2.59
For tissue T2 mean difference value is (10 - 105) / 73.3 = -1.26

Value (and absolute value) is much bigger in T1, hence gene might be a SVG which is highly expressed in tissue T1.

Algorithm: Mean difference value

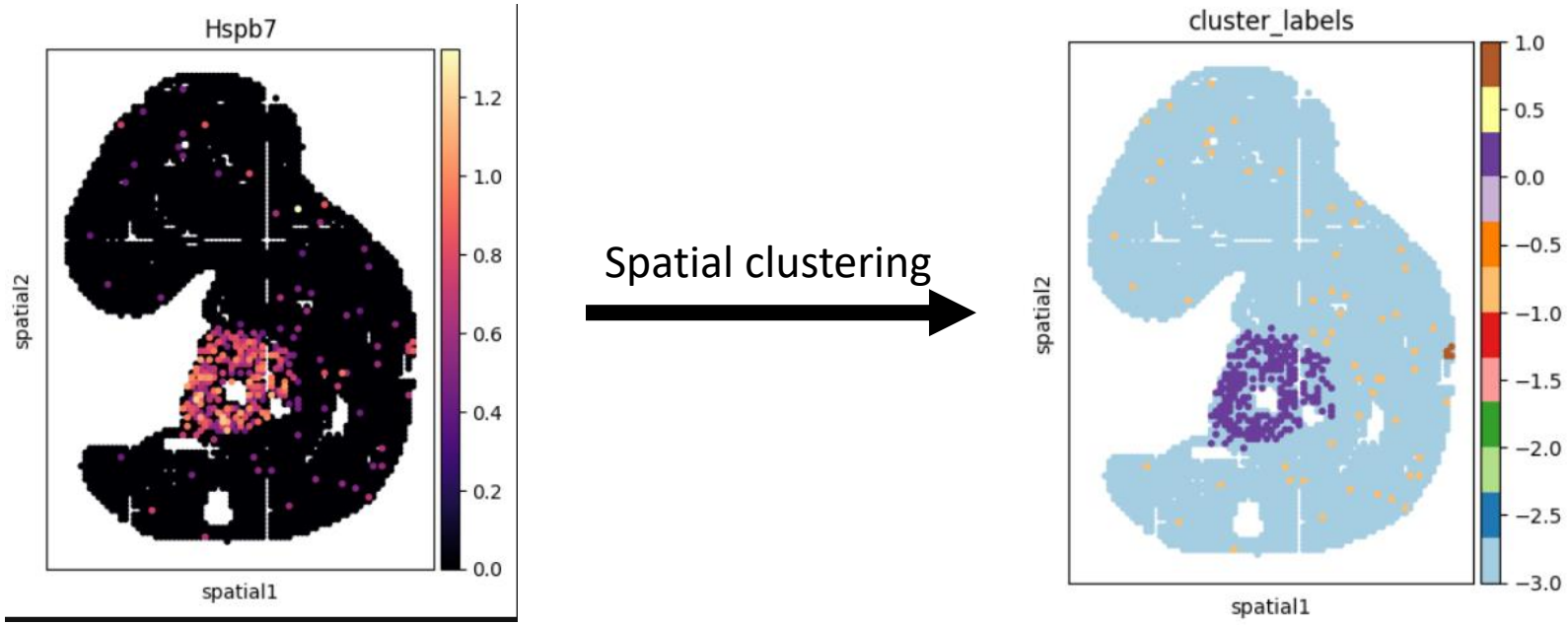
- Implementation:

```
def average_metric(self, gene_name):

    gene_data = self.X[:, self.var_names == gene_name]

    gene_data = gene_data / gene_data.max()  # Normalizing gene expressions

    # Calcululate difference of means in tissue and outside of tissue for all tissues
    differences = np.empty(len(self.tissue_names))
    for tissue_index, tissue_name in enumerate(self.tissue_names):


        mean_inside_tissue = gene_data[self.annotations == tissue_name].mean()
        mean_outside_tissue = gene_data[self.annotations != tissue_name].mean()
        differences[tissue_index] = mean_inside_tissue - mean_outside_tissue


    # We find the maximum difference between tissue and not tissue means and normalize it
    maximum_normalized_mean_difference = np.max(differences) # gene_mean_organism

    return maximum_normalized_mean_difference
```

Algorithm: Spatial clustering

This algorithm uses the fact that if a **sparse** gene (low number of cells) is a SVG its cells are spatially grouped. We can find these groups of nearby cells by clustering cells based on their spatial coordinates.

Groups of cells which are close together are grouped into clusters, while standalone cells are declared as noise. If there are enough cells in a cluster we can claim that gene is a SVG. Furthermore, if there are clusters in most of the tissues we dismiss this gene as it is present in most of the organism.



Genes in heart region have been grouped in purple cluster and the size of this cluster is sufficient to declare this gene a SVG. There is another brown cluster in the Mesenchyme region but its size is to small and it is filtered.

Algorithm: Spatial clustering

- Implementation:

```
def clustering_metric(self, gene, gene_data, gene_expressed_vector):

    gene_data = self.adata[:, self.var_names == gene]
    gene_X = self.X[:, self.var_names == gene]

    gene_expressed_vector = (gene_X > 0).flatten()
    gene_cell_num = gene_expressed_vector.sum()
    gene_clustering_data = gene_data[gene_expressed_vector] # Only cells with expressed gene are clustered

    clustering = DBSCAN(eps=2, min_samples=4).fit(gene_clustering_data.obsm['spatial'])
    labels = clustering.labels_
    cluster_ids = set(labels)

    # For each cluster
    all_tissues = set()
    for cluster_id in cluster_ids:

        if cluster_id == -1:  # Skipping noise
            continue

        cluster_cells = (labels == cluster_id).sum()  # Calculating number of cells

        # If cluster is big enough see in how many tissues it spans
        if cluster_size > 30:
            tissues_in_cluster = set(gene_clustering_data[cluster_cells].obs['annotation'])
            all_tissues.update(tissues_in_cluster)

    # If there are big cluster in at least 1 tissue but not at too many (6) we declare gene as SVG
    if 1 <= len(all_tissues) <= 6:
        return True
    else:
        return False
```

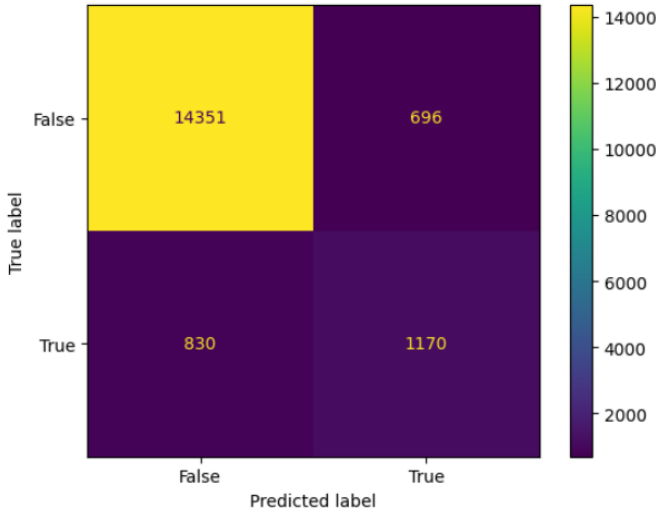
Algorithm: Combination

Mean difference value algorithm has shown to work well on genes with high number of cells, but with lower number of cells it can be very susceptible to noise and small differences in values between tissues. On the other hand, clustering algorithm works only with low number of cells. We can use both algorithms together by using mean difference value on genes with high number of cells and clustering on genes with low number of cells.

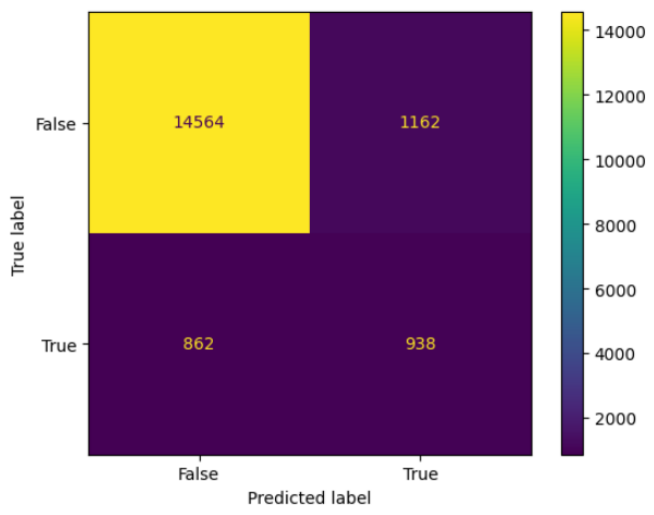
- Results:

	Mouse embrio	Mouse brain
F1 score	0.605	0.481
AUC	0.769	0.723

Confusion matrix for embrio



Confusion matrix for brain



Algorithm: Variance of entropy values

The idea of this algorithm lies in the fact that SVGs are the genes with non-uniform expression across spatial coordinates. Taking into account that uniform probability yields maximum uncertainty and, therefore, maximum entropy, we have calculated the entropy of gene expression in every tissue.
If the variance of these values is high, it means that there is a difference in the distribution of gene expression across tissues. In other words, this gene is likely to be an SVG.

Implementation:

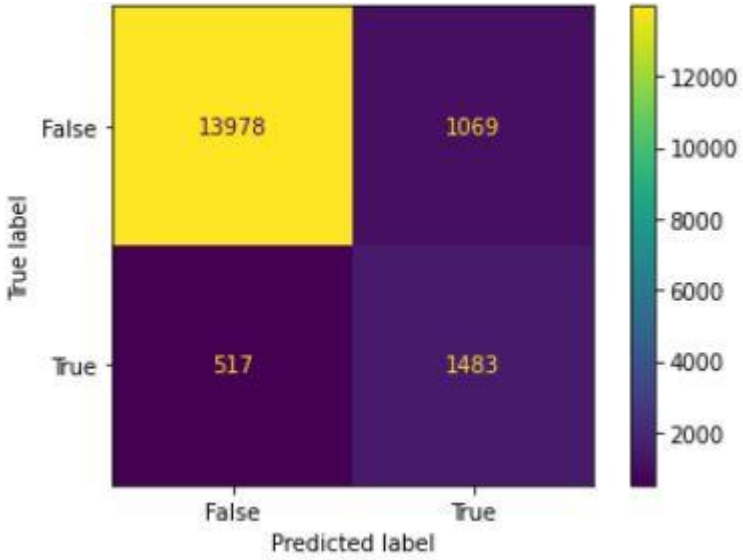
```
def varOfEntropy_metric(self, gene_name):

    gene_data = self.X[:, self.adata.var_names == gene_name]

    # Calculate the entropy of gene expression in every tissue
    entropy_values = []
    for i, c in enumerate(self.tissue_names):
        data = gene_data[self.adata.obs['annotation'] == c][:, None]
        # Calculate the histogram
        hist_values, _ = np.histogram(data, bins='auto')
        # Normalize the histogram to obtain the probability distribution
        pdf = hist_values / np.sum(hist_values)

        entropy_values.append(entropy(pdf))

    # Return the variance of entropy values
    return np.var(entropy_values)
```

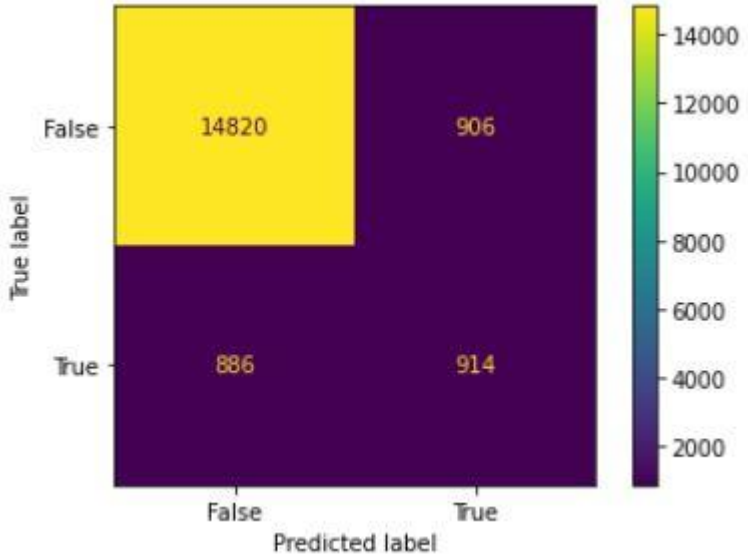
Entropy:

$$H(X) = -\sum_{x \in X} p(x) \log_b p(x)$$

Algorithm: Variance of entropy values

Results:





	Mouse embryo	Mouse brain
F1 score	65.16%	50.50%
AUC	83.52%	72.50%

Algorithm: Variance of mean values

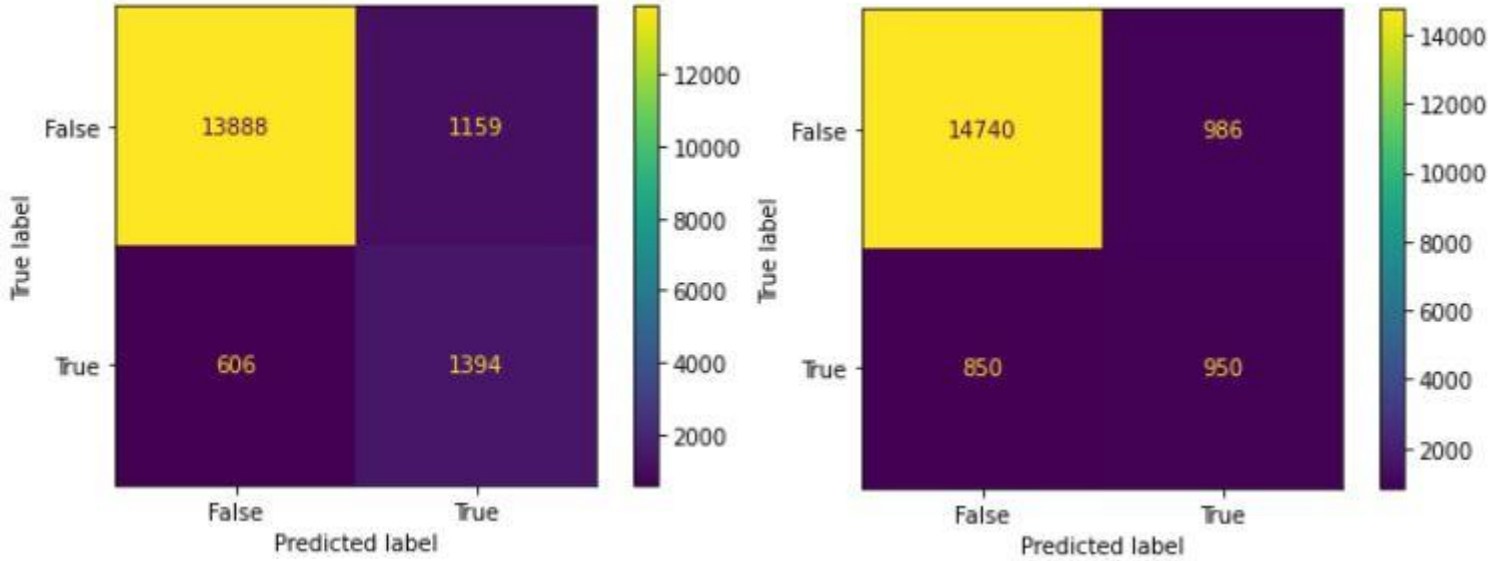
The idea of this algorithm was to calculate the mean of gene expression in every tissue and the variance of these values. If the variance of these values is high, it means that there is a difference in the level of expression across tissues, indicating that this gene is likely to be a SVG.

Implementation:

```
def varOfMean_metric(self, gene_name):

    gene_data = self.X[:, self.adata.var_names == gene_name]

    # Calculate the mean of gene expression in every tissue
    mean_values = []
    for i, c in enumerate(self.tissue_names):
        data = gene_data[self.adata.obs['annotation'] == c][:, None]
        mean_values.append(np.mean(data))

    # Return the variance of mean values
    return np.var(mean_values)
```

Algorithm: Variance of mean values

Results:



	Mouse embryo	Mouse brain
F1 score	61.23%	50.86%
AUC	80.99%	73.25%

Algorithm: Moran’s I

We have tried one popular approach for the identification of spatially variable genes, which is the Moran's I score. The Moran's I score is a measure of spatial autocorrelation, which quantifies the correlation of signals, such as gene expression, among observations that are close in space.

It is defines as:

$$I = \frac{n}{W} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} z_i z_y}{\sum_{i=1}^{n} z_i^2}$$

where

- z_i is the deviation of the feature from the mean $(x_i - \bar{X})$
- $w_{i,j}$ is the spatial weight between observations
- n is the number of spatial units
- W is the sum of all $w_{i,j}$

Algorithm: Moran's I

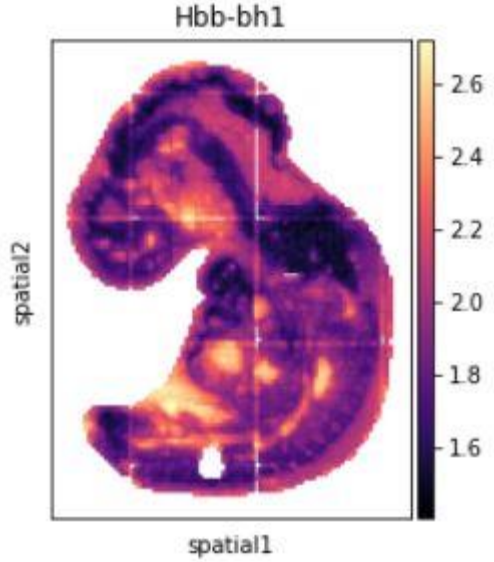
Implementation:

- It can be computed with Squidpy in Python with **squidpy.gr.spatial_autocorr()** and **mode = 'moran'**, but we first need to compute a spatial graph with **squidpy.gr.spatial_neighbors()**

```
sq.gr.spatial_neighbors(adata)
sq.gr.spatial_autocorr(adata, mode="moran", genes=adata.var_names)
```

- The method adds a dataframe to adata.uns under the key moranI.

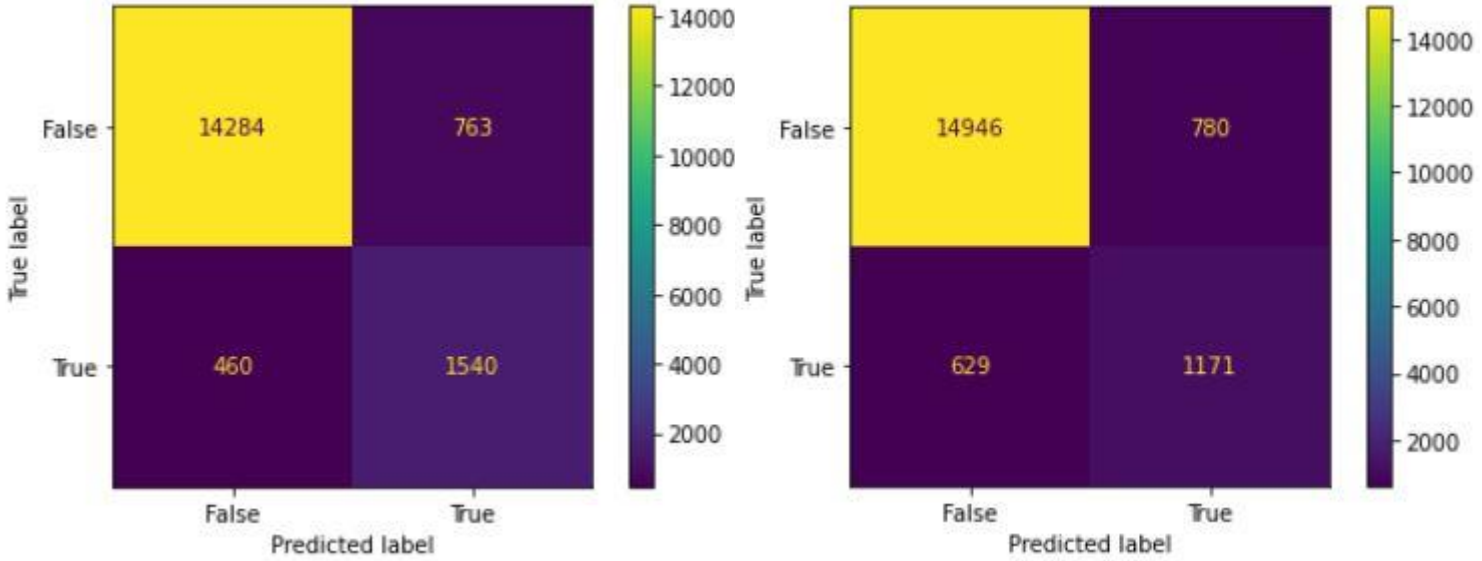
	I	pval_norm	var_norm	pval_norm_fdr_bh
Hbb-bh1	0.861195	0.000000	0.000057	0.000000



Tutorial

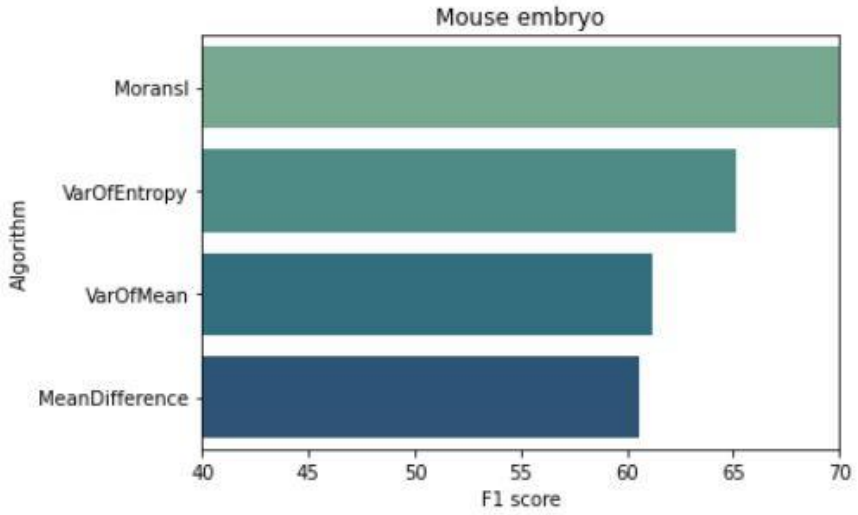
Algorithm: Moran's I

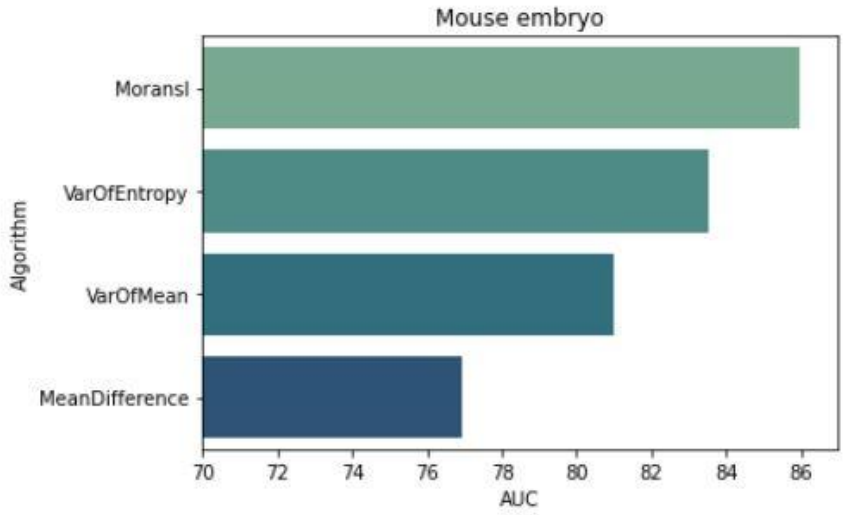
Results:

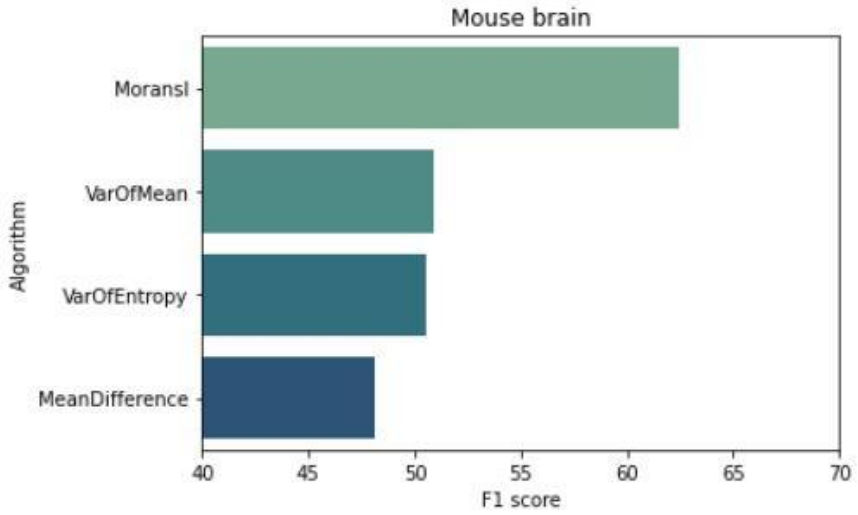


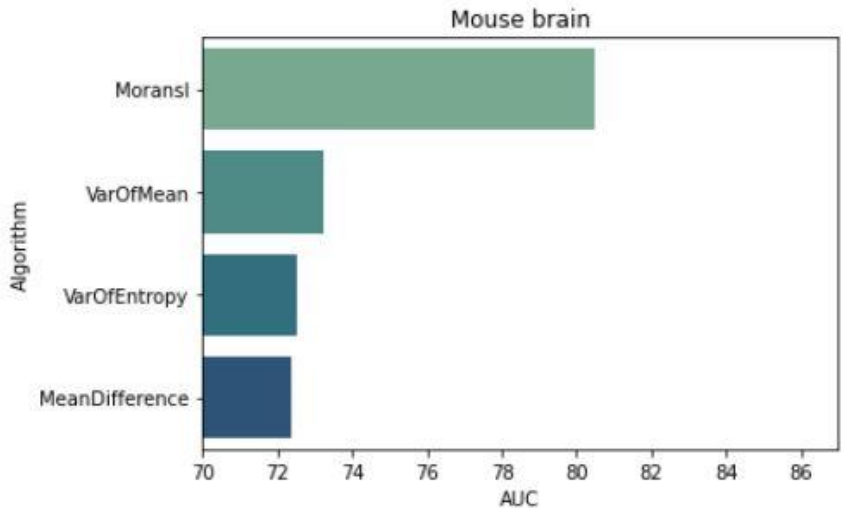
	Mouse embryo	Mouse brain
F1 score	71.58%	62.43%
AUC	85.96%	80.48%

Comparative Overview of Algorithm Results









Thank you for your attention!