A Appendix: experimental details

Probabilistic graph interpolation of traffic data

Here we consider the problem of interpolating traffic congestion data over a road network, consisting of San Jose highways. We build the road network, a weighted graph, from OpenStreetMap data (OpenStreetMap contributors, 2017) obtained using the OSMnx library (Boeing, 2017).

We obtain traffic congestion data from the *California Performance Measurement Systems* database (Chen et al., 2001). This system maps sensor location and time & date to (among other factors) traffic flow speed in miles per hour measured by the given sensor at a particular time and date. We chose the specific date and time to be 17:30 on Monday, the 2^{th} of January, 2017. We bind the traffic congestion data to the graph by adding additional nodes that subdivide existing edges of the graph at the location of the measurement points. The edge weights are assigned to be equal to the inverse travel distances between the corresponding nodes. When we encounter two nodes connected by multiple edges, we remove all but one edge between them, and set its weight to be the inverse mean travel distance between the pair of nodes. After this, the graph is processed by (1) twice subsequently removing hanging nodes, and (2) removing two other nodes located particularly far away from data. We end up with 325 labeled nodes on a sparse connected graph with 1016 nodes and 1173 edges.

For the model, we employ a GP with a graph Matérn kernel and the graph diffusion kernel. Both kernels are approximated using 500 eigenpairs of the graph Laplacian. The eigenpairs required to build graph kernels are computed using TensorFlow's eigenvalue decomposition (TF.LINALG.EIGH) routine and are thus exact up to a numerical error arising from the floating point arithmetic. We train by optimzing the kernel hyperparameters κ (length scale), σ^2 (variance) and for the case of the graph Matérn kernel, ν (smoothness), as well as the Gaussian likelihood variance ς^2. We use the following values for initialization: $\nu = 3/2$ for the graph Matérn kernel, $\kappa = 3$, $\sigma^2 = 1$, $\varsigma^2 = 0.01$. We randomly choose 250 nodes with traffic flow rates as training data. The likelihood is optimized for 20000 iterations with the ADAM optimizer, with learning rate set to the TensorFlow default value of 0.001.

We repeat the experiment 10 times. The average MSE over the 10 experiments and its standard deviation are presented in Table 1 for the graph Matérn kernel and for the graph diffusion kernel. The smoothness parameter ν after optimization varies across the runs. Its mean is 1.8 and the standard deviation is 0.9. One of these ten runs is visualized in Figures 3 and 4.

Multi-class classification in a scientific citation network

Here we consider multi-class classification on the largest connected component of the *Cora* citation network.

There are 2708 nodes corresponding to scientific publications in the *Cora* citation network. Usually, each node of this graph is attributed by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary of 1433 unique words in the abstract—we discard this portion of data to focus on other problem aspects.[4] Two nodes are connected by an edge if one paper cites another. The edges are unweighted and undirected, and there are 5429 edges overall. The nodes are labeled into one of seven classes. We concentrate on the largest connected component of the Cora graph, which has 2485 nodes and 5069 edges.

We consider a Gaussian process $V \to \mathbb{R}^7$ with independent components as a latent process for the classification task. The process is then combined with the robust max function $\boldsymbol{\varphi}(\cdot)$ to yield a vector of probabilities over the classes. Let $\boldsymbol{f} = (f_1, .., f_7)$ denote the values of 7-dimensional latent process. Then we have

$$\boldsymbol{\varphi}(\boldsymbol{f}) = (\varphi_1(\boldsymbol{f}), .., \varphi_7(\boldsymbol{f})) \qquad \varphi_c(\boldsymbol{f}) = \begin{cases} 1 - \varepsilon, & \text{if } c = \arg\max_c f_c \\ \varepsilon/6, & \text{otherwise,} \end{cases} \tag{24}$$

where ε is fixed to be 10^{-3}. For the likelihood, we employ the categorical distribution. We use the graph Matérn kernel and the graph diffusion kernel for the prior. Both kernels are approximated using 500 eigenpairs of the graph

[4] It's also possible to use a separable kernel given by the product of a graph Matérn kernel and a Euclidean kernel operating on word vectors, but this model is unlikely to perform well due to usual GP difficulties in high dimension. Incorporating this portion of the data effectively therefore necessitates additional modeling considerations, so we do not focus on it here.

Laplacian. The eigenpairs required are computed using TensorFlow eigenvalue decomposition (TF.LINALG.EIGH) and are thus exact up to a numerical error arising from use of floating point arithmetic. We use 140 random nodes as a training set and predict labels for the remaining 2345 nodes.

For the variational GP, we employ the standard SVGP model of GPflow (Matthews et al., 2017; van der Wilk et al., 2020). We fix the the coordinates of inducing points, which are located on the graph, to the data locations, and do not optimize them. We restrict the covariance of the inducing distribution to a diagonal form, since this results in the best performance in this setting. The inducing mean is initialized to be zero and the inducing covariance is initialized to be the identity. The prior hyperparameters are initialized to be $\nu = 3$ for the Matérn kernel, $\kappa = 5$, $\sigma^2 = 1$. The variational objective is optimized for 20000 iterations with the ADAM optimizer whose learning rate set to 0.001. Convergence is usually achieved in a significantly lower number of iterations. The SVGP batch size is equal to the number of training data points, and whitening is enabled.

We repeat the experiment 10 times. Each time, we additionally randomly choose a test set of 1000 nodes. The average (over the 10 experiments) accuracy (fraction of the correctly classified points) and its standard deviation are presented in Table 1 for the graph Matérn kernel and for the graph diffusion kernel. The value of the smoothness parameter ν after optimization varies considerably across runs. The mean ν is 3.2 and the standard deviation is 2.7. The results of one of said ten runs are visualized on Figures 6, 7, 8. Note that Figures 6 and 8 repeat Figure 5 (a), (b) at a higher resolution.

B Appendix: graph Fourier features

Assume that $\boldsymbol{\Delta} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$ with $\mathbf{U}$ orthogonal and $\boldsymbol{\Lambda}$ diagonal. For a set of of locations $\boldsymbol{x} \in V^n$ define $\mathbf{U}(\boldsymbol{x})$ to be the $n \times d$ submatrix of $\mathbf{U}$ with rows corresponding to the elements of $\boldsymbol{x}$. Define also $\Psi(\lambda) = \Phi(\lambda)^{-2}$, where Φ is given in (10). Then, if f is a graph Matérn process and $\mathbf{K}_{\boldsymbol{xx}}$ is the covariance matrix of $f(\boldsymbol{x})$, we have

$$\mathbf{K}_{\boldsymbol{xx}} = \mathbf{U}(\boldsymbol{x})\Psi(\boldsymbol{\Lambda})\mathbf{U}(\boldsymbol{x})^T \approx \widetilde{\mathbf{U}}(\boldsymbol{x})\Psi(\widetilde{\boldsymbol{\Lambda}})\widetilde{\mathbf{U}}(\boldsymbol{x})^T, \tag{25}$$

where $\widetilde{\mathbf{U}}(\boldsymbol{x})$ and $\widetilde{\boldsymbol{\Lambda}}$ are the approximations of $\widetilde{\mathbf{U}}(\boldsymbol{x})$ and $\widetilde{\boldsymbol{\Lambda}}$ with elements in the last $d - l$ rows of matrices $\widetilde{\mathbf{U}}(\boldsymbol{x})$ and $\widetilde{\boldsymbol{\Lambda}}$ set to zero. This approximation arises from finding only the l smallest eigenpairs, which can be done more efficiently than the full eigendecomposition by employing, for instance, the Lanczos algorithm.

With this, using the Woodbury matrix identity, we get

$$\big(\mathbf{K}_{\boldsymbol{xx}} + \sigma^2\mathbf{I}\big)^{-1} = \sigma^{-2}\mathbf{I} - \sigma^{-4}\mathbf{U}(\boldsymbol{x})\big(\Psi(\boldsymbol{\Lambda})^{-1} + \overbrace{\sigma^{-2}\mathbf{U}(\boldsymbol{x})^T\mathbf{U}(\boldsymbol{x})}^{\sigma^{-2}\mathbf{I}}\big)^{-1}\mathbf{U}(\boldsymbol{x})^T, \tag{26}$$

$$\big(\mathbf{K}_{\boldsymbol{xx}} + \sigma^2\mathbf{I}\big)^{-1} \approx \sigma^{-2}\mathbf{I} - \sigma^{-4}\widetilde{\mathbf{U}}(\boldsymbol{x})\big(\Psi(\widetilde{\boldsymbol{\Lambda}})^{-1} + \underbrace{\sigma^{-2}\widetilde{\mathbf{U}}(\boldsymbol{x})^T\widetilde{\mathbf{U}}(\boldsymbol{x})}_{\text{diagonal}}\big)^{-1}\widetilde{\mathbf{U}}(\boldsymbol{x})^T, \tag{27}$$

where a diagonal matrix is inside the brackets. It is clear that computing $\big(\mathbf{K}_{\boldsymbol{xx}} + \sigma^2\mathbf{I}\big)^{-1}\boldsymbol{y}$ when the (approximation of) eigenpairs are known can be done without incurring the $O(n^3)$ cost, resulting in efficient computation.

The resulting kernel can be (approximately) expressed by

$$k(i,j) = \sum_{s=1}^{d} \Psi(\lambda_s)\boldsymbol{u}_{\boldsymbol{s}}(i)\boldsymbol{u}_{\boldsymbol{s}}(j) \approx \sum_{s=1}^{l} \Psi(\lambda_s)\boldsymbol{u}_{\boldsymbol{s}}(i)\boldsymbol{u}_{\boldsymbol{s}}(j), \tag{28}$$

where λ_s are eigenvalues of $\boldsymbol{\Delta}$, while $\boldsymbol{u}_{\boldsymbol{s}}(i)$ and $\boldsymbol{u}_{\boldsymbol{s}}(j)$ are the i-th and j-th component of the eigenvector $\boldsymbol{u}_{\boldsymbol{s}}$ corresponding to λ_s. For Matérn kernels we have $\Psi(\lambda) = \big(\frac{2\nu}{\kappa^2} + \lambda\big)^{-\nu}$ from which it is clear that differentiating k with respect to the smoothness parameter ν is straightforward. This allows to optimize the marginal likelihood over the smoothness parameter ν in the same way this optimization over the length scale parameter κ is usually done.

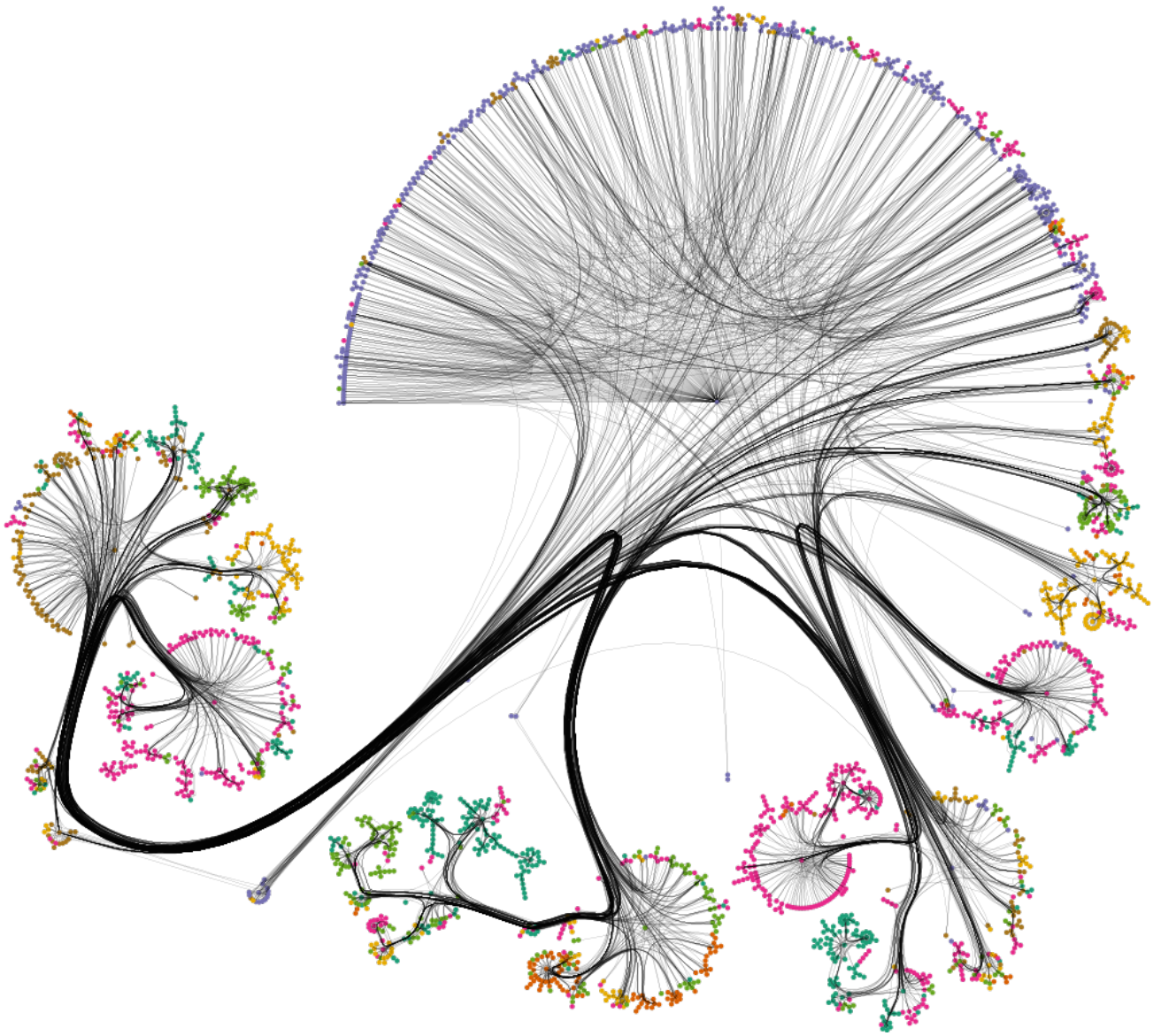

Figure 6: Topic classification for the *Cora* citation network, using the citation graph. Ground truth.

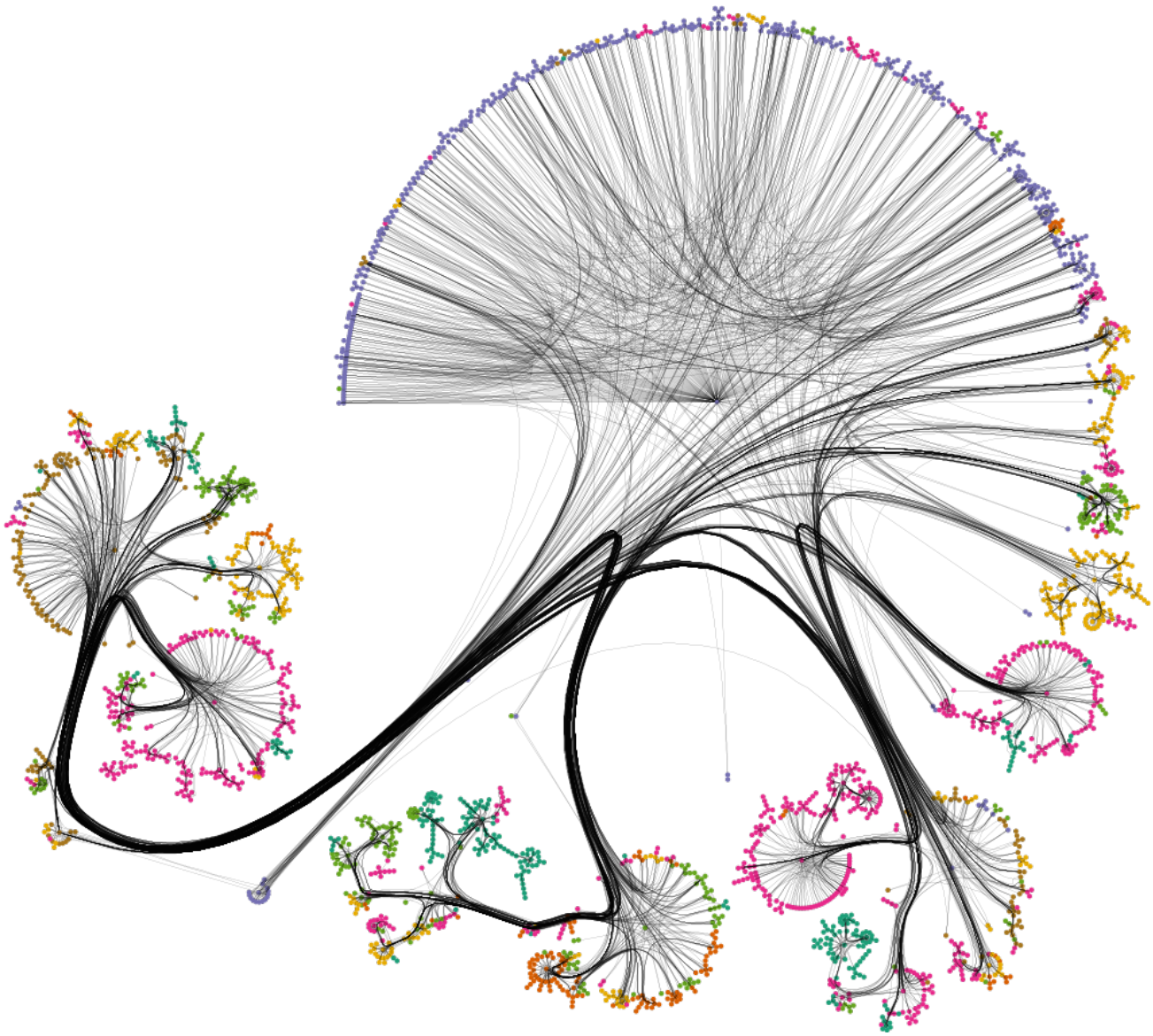

Figure 7: Topic classification for the *Cora* citation network, using the citation graph. Prediction.

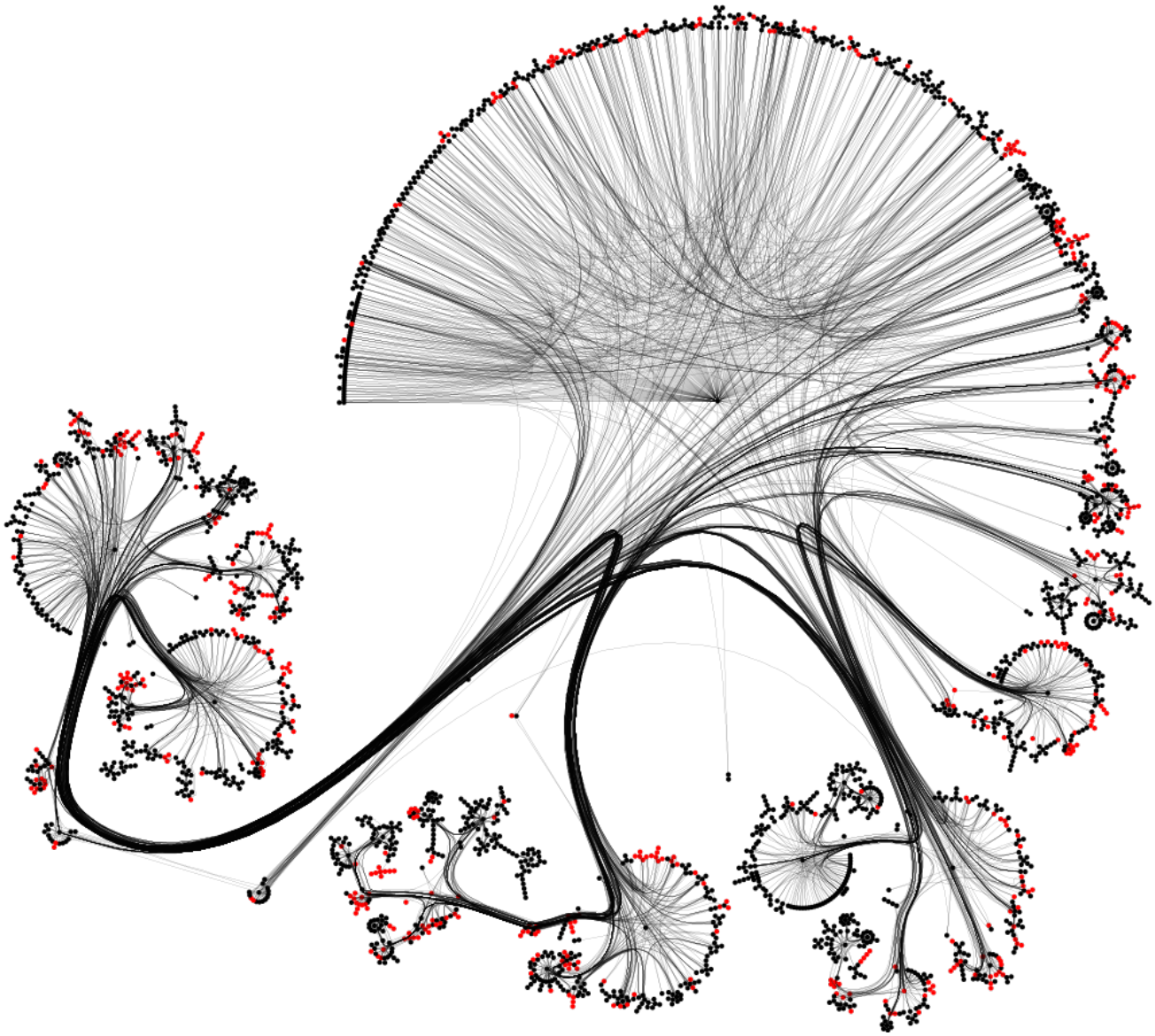

Figure 8: Topic classification for the *Cora* citation network, using the citation graph. Prediction errors. Red nodes are misclassified.