Appendix A. Detailed Derivations and Proofs for Sec. 3.1 & 3.2

A.1 Details for Sec. 3.1

To better present our proposed vMF classifier, we formulate Eq. 2 in submission PDF equivalently as:

\[ p^l_i = \frac{p_{D}^{l,\text{a}}(i) \cdot p(\tilde{x}^l|\kappa_i, \tilde{\mu}_i)}{\sum_{j=1}^{C} p_{D}^{l,\text{a}}(j) \cdot p(\tilde{x}^l|\kappa_j, \tilde{\mu}_j)} \]

\[ = \frac{\exp\left\{ \kappa_i \cdot \tilde{x}^l \tilde{\mu}_i^\top + \left(\frac{d}{2} - 1\right) \cdot \log \kappa_i - \log I_{d-1}(\kappa_i) + \log b_i \right\}}{\sum_{j=1}^{C} \exp\left\{ \kappa_j \cdot \tilde{x}^l \tilde{\mu}_j^\top + \left(\frac{d}{2} - 1\right) \cdot \log \kappa_j - \log I_{d-1}(\kappa_j) + \log b_j \right\}} \]

where \( b_i \) is denoted as \( b_i \) and \( b_j \) is denoted as \( b_j \).

Based on Eq. (1) in Appendix, we calculate the derivative of \( p^l_i \) with respect to \( \kappa_i \) as:

\[ \frac{\partial p^l_i}{\partial \kappa_i} = \frac{\partial p^l_i}{\partial (\kappa_i \cdot \tilde{x}^l \tilde{\mu}_i^\top + b_i)} \cdot \left( \frac{\partial (\kappa_i \cdot \tilde{x}^l \tilde{\mu}_i^\top)}{\partial \kappa_i} + \frac{\partial b_i}{\partial \kappa_i} \right) \]

\[ = \left(1 - p^l_i \right) \cdot (\tilde{x}^l \tilde{\mu}_i^\top - A_d(\kappa_i)) \]

where \( A_d(\kappa_i) = I_{d/2}(\kappa_i)/I_{d/2-1}(\kappa_i) \). The derivative of \( p^l_i \) with respect to \( \kappa_j \) is calculated as:

\[ \frac{\partial p^l_i}{\partial \kappa_j} = \frac{\partial p^l_i}{\partial (\kappa_j \cdot \tilde{x}^l \tilde{\mu}_j^\top + b_j)} \cdot \left( \frac{\partial (\kappa_j \cdot \tilde{x}^l \tilde{\mu}_j^\top)}{\partial \kappa_j} + \frac{\partial b_j}{\partial \kappa_j} \right) \]

\[ = -p^l_j \cdot (\tilde{x}^l \tilde{\mu}_j^\top - A_d(\kappa_j)) \]

The derivatives of \( p^l_i \) with respect to \( \tilde{\mu}_i \) and \( \tilde{\mu}_j \) are formulated as:

\[ \frac{\partial p^l_i}{\partial \tilde{\mu}_i} = \frac{\partial p^l_i}{\partial (\kappa_i \cdot \tilde{x}^l \tilde{\mu}_i^\top)} \cdot \frac{\partial (\kappa_i \cdot \tilde{x}^l \tilde{\mu}_i^\top)}{\partial \tilde{\mu}_i} = (1 - p^l_i) \cdot \kappa_i \cdot \tilde{x}^l \]

\[ \frac{\partial p^l_i}{\partial \tilde{\mu}_j} = \frac{\partial p^l_i}{\partial (\kappa_j \cdot \tilde{x}^l \tilde{\mu}_j^\top)} \cdot \frac{\partial (\kappa_j \cdot \tilde{x}^l \tilde{\mu}_j^\top)}{\partial \tilde{\mu}_j} = -p^l_j \cdot \kappa_j \cdot \tilde{x}^l \]

Implement Details. Both forward and backward operations with respect to \( b_i \) are not supported by Pytorch framework. In addition, the floating point precision for \( I_v(\kappa) \) with the large \( v \) and small \( \kappa \) (e.g., \( v = 511 \) and \( \kappa = 16 \)) exceeds float64 which is the maximum floating point precision of CUDA. While the floating point precision for \( \log I_v(\kappa) \) is in the normal interval.

To implement our method, we first calculate \( b_i \) and its derivative by mpmath library which allows the floating pointing operation with arbitrary precision. Then, we convert them to the data type of Pytorch. Here is our core code for the above steps:
import mpmath as mp
import numpy as np
import torch

Iv = np.frompyfunc(mp.besseli, 2, 1)  # Bessel Function I
log = np.frompyfunc(mp.log, 1, 1)  # Logarithmic Function

# Forward and backward functions for b_i

class Function_Bias(torch.autograd.Function):
    @staticmethod
    def forward(self, d, kappa):
        self.k = kappa.data.cpu().numpy()
        self.v = d / (2 - 1)
        bias = self.v * log(self.k) - log(Iv(self.v, self.k))
        return torch.Tensor([float(bias)]).type_as(kappa)

    return bias

@staticmethod
    def backward(self, grad_output):
        kappa = self.saved_tensors[-1]
        Adk = Iv(self.v + 1, self.k) / Iv(self.v, self.k)
        return None, -grad_output * Adk

See core code in supplement material for more implement details.

A.2 Details for Sec. 3.2

For simplification, we abbreviate $o_{\Lambda}(\kappa_i, \kappa_j, \tilde{\mu}_i, \tilde{\mu}_j)$ as $o_{\Lambda}$. To better present the distribution overlap coefficient, we formulate Eq. 6 in submission PDF equivalently as:

$$KL_{ij} = \ln \frac{C_d(\kappa_i)}{C_d(\kappa_j)} + A_d(\kappa_i) \cdot (\kappa_i - \kappa_j \cdot \tilde{\mu}_i \tilde{\mu}_j^\top)$$

$$= b_i - b_j + A_d(\kappa_i) \cdot (\kappa_i - \kappa_j \cdot \tilde{\mu}_i \tilde{\mu}_j^\top).$$

The derivatives of $o_{\Lambda}$ with respect to $\kappa_i$ and $\kappa_j$ are formulated as:

$$\frac{\partial o_{\Lambda}}{\partial \kappa_i} = \frac{\partial o_{\Lambda}}{\partial KL_{ij}} \cdot \frac{\partial KL_{ij}}{\partial \kappa_i} = o_{\Lambda}^2 \cdot \frac{\partial A_d(\kappa_i)}{\partial \kappa_i} \cdot (-\kappa_i + \kappa_j \cdot \tilde{\mu}_i \tilde{\mu}_j^\top)$$

$$\frac{\partial o_{\Lambda}}{\partial \kappa_j} = \frac{\partial o_{\Lambda}}{\partial KL_{ij}} \cdot \frac{\partial KL_{ij}}{\partial \kappa_j} = o_{\Lambda}^2 \cdot (-A_d(\kappa_j) + A_d(\kappa_i) \cdot \tilde{\mu}_i \tilde{\mu}_j^\top),$$

where the derivative of $A_d(\kappa_i)$ with respect to $\kappa_i$ is defined as:

$$\frac{\partial A_d(\kappa_i)}{\partial \kappa_i} = 1 - \frac{d - 1}{\kappa_i} \cdot A_d(\kappa_i) - A_d^2(\kappa_i).$$

The derivatives with respect to $\tilde{\mu}_i$ and $\tilde{\mu}_j$ are easily derived, following the above operations. The results are demonstrated in Tab. 1 of the submission PDF. 

Implement Details. Facing the same case as $b_i$, we need to define the forward and backward functions of $A_d(\kappa_i)$ manually. The core code is demonstrated as:
import mpmath as mp
import numpy as np
import torch

Iv = np.frompyfunc(mp.besselI, 2, 1)  # Bessel Function I_v()

# Forward and backward functions for A_d(\kappa)
class Function_Adk(torch.autograd.Function):
    @staticmethod
    def forward(self, d, kappa):
        k = kappa.data.cpu().numpy()
        self.d, v = d, d / 2 - 1
        Adk = Iv(v+1, k) / Iv(v, k)
        Adk = torch.Tensor([float(Adk)]).type_as(kappa)
        self.save_for_backward(kappa, Adk)
        return Adk
    @staticmethod
    def backward(self, grad_output):
        kappa, Adk = self.saved_tensors
        grad_Adk = (self.d - 1) / kappa * Adk - Adk ** 2
        return None, grad_output * grad_Adk

Appendix B. Relation with Other classifiers

B.1 Balanced Cosine Classifier

Setting \( \kappa_i = \text{const} \sigma, \forall i \in [1, C] \), Eq. 2 in submission PDF can be re-write as:

\[
p_i = \frac{p_D^{\text{tra}}(i) \cdot p(\tilde{x} | \sigma, \tilde{\mu}_i)}{\sum_{j=1}^C p_D^{\text{tra}}(j) \cdot p(\tilde{x} | \sigma, \tilde{\mu}_j)} = \frac{n_i \cdot \exp\{\sigma \cdot \tilde{x}^\top \tilde{\mu}_i\}}{\sum_{j=1}^C n_j \cdot \exp\{\sigma \cdot \tilde{x}^\top \tilde{\mu}_j\}}. \tag{8}
\]

Consequently, the balanced cosine classifier can be considered a special case of our vMF classifier.

B.2 Convert Other Classifiers into Ours

In this paper, we take three classifiers into account, including linear, \( \tau \)-norm, and causal classifiers, following the default setting (i.e., ignoring the bias terms). To measure the distribution overlap coefficient of them above, we develop a conversion method to convert them in a vMF classifier way.

For the linear classifier, given a feature vector \( \mathbf{x} \in \mathbb{R}^{1 \times d} \) and classifier weights \( W^{lin} = \{w_1^{lin}, ..., w_i^{lin}, ..., w_C^{lin}\} \in \mathbb{R}^{C \times d} \), the score for class \( i \) can be defined as:

\[
s_i^{lin} = w_i^{lin} \mathbf{x}^\top = \underbrace{\|w_i^{lin}\|_2}_{\text{compactness}} \cdot \underbrace{\mathbf{x}^\top}_{\text{orientation}}. \tag{9}
\]
For the $\tau$-norm classifier, given a feature vector $\mathbf{x} \in \mathbb{R}^{1 \times d}$ and classifier weights $\mathbf{W}^\tau = \{ \mathbf{w}_1^\tau, \ldots, \mathbf{w}_i^\tau, \ldots, \mathbf{w}_C^\tau \} \in \mathbb{R}^{C \times d}$, the score for class $i$ can be defined as:

$$s_i^\tau = \frac{\mathbf{w}_i^\tau}{\| \mathbf{w}_i^\tau \|_2} \cdot \mathbf{x}^\top = \underbrace{\frac{\| \mathbf{w}_i^\tau \|_2}{\| \mathbf{w}_i^\tau \|_2} \cdot \mathbf{x}^\top}_{\text{orientation}} \times \underbrace{\mathbf{w}_i^\tau}_{\text{compactness}}.$$ (10)

For the causal classifier, given a feature vector $\mathbf{x} \in \mathbb{R}^{1 \times d}$ and classifier weights $\mathbf{W}^{cau} = \{ \mathbf{w}_1^{cau}, \ldots, \mathbf{w}_i^{cau}, \ldots, \mathbf{w}_C^{cau} \} \in \mathbb{R}^{C \times d}$, the score for class $i$ can be defined as:

$$s_i^{cau} = \frac{\mathbf{w}_i^{cau}}{\| \mathbf{w}_i^{cau} \|_2 + \gamma} \cdot \mathbf{x}^\top = \underbrace{\frac{\| \mathbf{w}_i^{cau} \|_2 + \gamma}{\| \mathbf{w}_i^{cau} \|_2 + \gamma}}_{\text{orientation}} \cdot \underbrace{\mathbf{w}_i^{cau}}_{\text{compactness}} \cdot \mathbf{x}^\top.$$ (11)

In our experiment, the optimal setting $\tau$ of $\tau$-norm classifier [2] is equal to 0.7. $\gamma$ of the causal classifier [4] is set as 1/16, following the official codes. For the causal classifier, we do not apply the causal post-processing algorithm proposed by them. In addition, our ablation study on post-training calibration algorithm with different classifiers is shown in Tab. 4 of submission PDF.

References