使用 Hofstede 文化维度对不同国家文化的聚类和可视化

实验目的

本实验旨在利用 Hofstede 文化维度理论，对不同国家的文化特征进行量化分析，并通过主成分分析（PCA）降维和 KMeans 聚类算法，对这些国家的文化进行聚类和可视化。

实验数据来自 https://geerthofstede.com/research-and-vsm/dimension-data-matrix/ 。

实验原理

Hofstede 文化维度理论

Hofstede 文化维度理论是跨文化研究中的重要理论框架，它通过六个维度来描述不同国家的文化特征：

1. **权力距离（Power Distance, PDI）**：社会对权力不平等分布的接受程度。
2. **个人主义 vs 集体主义（Individualism vs Collectivism, IDV）**：个人与群体之间的关系倾向。
3. **男性气质 vs 女性气质（Masculinity vs Femininity, MAS）**：社会对竞争、成就和关怀的偏好。
4. **不确定性规避（Uncertainty Avoidance, UAI）**：社会对不确定性和模糊性的容忍程度。
5. **长期导向 vs 短期导向（Long-term vs Short-term Orientation, LTO）**：社会对长期规划与传统价值观的重视程度。
6. **放纵 vs 克制（Indulgence vs Restraint, IND）**：社会对满足欲望和享受生活的态度。

主成分分析（PCA）

PCA 是一种常用的降维技术，它通过线性变换将高维数据映射到低维空间，同时保留数据的主要特征。在本实验中，PCA 用于将六个文化维度降维到二维空间，以便于可视化和分析。

KMeans 聚类

KMeans 是一种无监督学习算法，用于将数据点划分为 K 个簇。它通过迭代优化簇中心，使得同一簇内的数据点尽可能相似，而不同簇之间的数据点尽可能不同。在本实验中，KMeans 用于对降维后的文化数据进行聚类，以识别具有相似文化特征的国家群体。

核心代码

为便于在 Web 端实现可视化，本实验主要采用 JavaScript/TypeScript 语言编写，使用了 Math.js 库。除使用前端框架 React 和 Chart.js 实现可视化外，算法实现和 Python 等其他语言并无太大差别。核心代码如下：

pca.ts

```ts
import {
  matrix,
  Matrix,
  multiply,
  transpose,
  eigs,
  MathArray,
  std,
  mean,
  subtract,
  dotDivide,
} from "mathjs";

export function pca(data: number[][], n: number) {
  // 将数据转换为矩阵
  const dataMatrix = matrix(data);
  // 计算均值
  const dataMean = mean(dataMatrix, 0);
  // 计算标准差
  const dataStd = std(dataMatrix, 0);
  // 标准化数据
  const standardlizedData = dotDivide(subtract(dataMatrix, dataMean), dataStd) as unknown
as Matrix;
  // 计算协方差矩阵
  const covarianceMatrix = multiply(transpose(standardlizedData), standardlizedData);
  // 计算特征向量
  const { eigenvectors } = eigs(covarianceMatrix);
  // 选择前 n 个特征向量
  const topEigenVectors = eigenvectors
    .sort(({ value: v1 }, { value: v2 }) => (v2 as number) - (v1 as number))
    .slice(0, n);
  const eigenMat = transpose(matrix(topEigenVectors.map(({ vector }) => vector as
MathArray)));
  // 将数据投影到新的低维空间
  const transformedData = multiply(standardlizedData, eigenMat);

  return transformedData.toArray() as number[][];
}
```

kmeans.ts

```ts
type Point = number[];

// 欧氏距离
function distance(a: Point, b: Point): number {
  return Math.sqrt(a.reduce((sum, val, i) => sum + Math.pow(val - b[i], 2), 0));
```

```
}

// KMeans++ 初始化中心点
function initCentroids(points: Point[], k: number): Point[] {
  // 随机选择一个点作为第一个中心点
  const centroids: Point[] = [points[Math.floor(Math.random() * points.length)]];

  // 继续选择剩余的中心点
  while (centroids.length < k) {
    // 计算每个点到最近中心点的距离
    const distances = points.map((point) => {
      return Math.min(...centroids.map((centroid) => distance(point, centroid)));
    });

    // 计算总距离
    const totalDistance = distances.reduce((sum, dist) => sum + dist, 0);

    // 计算每个点被选为下一个中心点的概率
    const probabilities = distances.map((dist) => dist / totalDistance);

    // 计算累积概率
    const cumulProbabilities = probabilities.map((_, i) => {
      return probabilities.slice(0, i + 1).reduce((sum, p) => sum + p, 0);
    });

    // 根据累积概率随机选择一个点作为下一个中心点
    const randomValue = Math.random();
    const selectedIndex = cumulProbabilities.findIndex((cp) => cp ≥ randomValue);

    centroids.push(points[selectedIndex]);
  }

  return centroids;
}

// 分配每个点到最近的中心点
function assignPoints(centroids: Point[], points: Point[]) {
  const clusters: Point[][] = Array.from(centroids, () => []);

  points.forEach((point) => {
    const [, clusterIndex] = centroids.reduce(
      ([minDist, clusterIndex], centroid, index) => {
        const dist = distance(point, centroid);
        if (dist < minDist) {
          return [dist, index];
        }
        return [minDist, clusterIndex];
      },
      [Infinity, 0]
    );
```

```
    clusters[clusterIndex].push(point);
  });

  return clusters;
}

function updateCentroids(centroids: Point[], clusters: Point[][]) {
  return clusters.map((cluster) => {
    if (cluster.length === 0) {
      // 如果簇为空，随机选择一个中心点
      return centroids[Math.floor(Math.random() * centroids.length)];
    }

    // 计算簇的均值
    const centroid = cluster[0].map((_, i) => {
      return cluster.reduce((sum, point) => sum + point[i], 0) / cluster.length;
    });

    return centroid;
  });
}

// KMeans 算法实现
export function kmeans(points: Point[], k: number, maxIterations: number = 100): Point[]
[] {
  let centroids = initCentroids(points, k);
  let clusters = Array.from({ length: k }, (): Point[] => []);

  for (let iter = 0; iter < maxIterations; iter++) {
    const updatedClusters = assignPoints(centroids, points);
    const updatedCentroids = updateCentroids(centroids, updatedClusters);

    centroids = updatedCentroids;
    clusters = updatedClusters;

    if (centroids.every((centroid, i) => distance(centroid, updatedCentroids[i]) < 1e-6))
{
      break;
    }
  }

  return clusters;
}
```

实验结果

PCA 降维

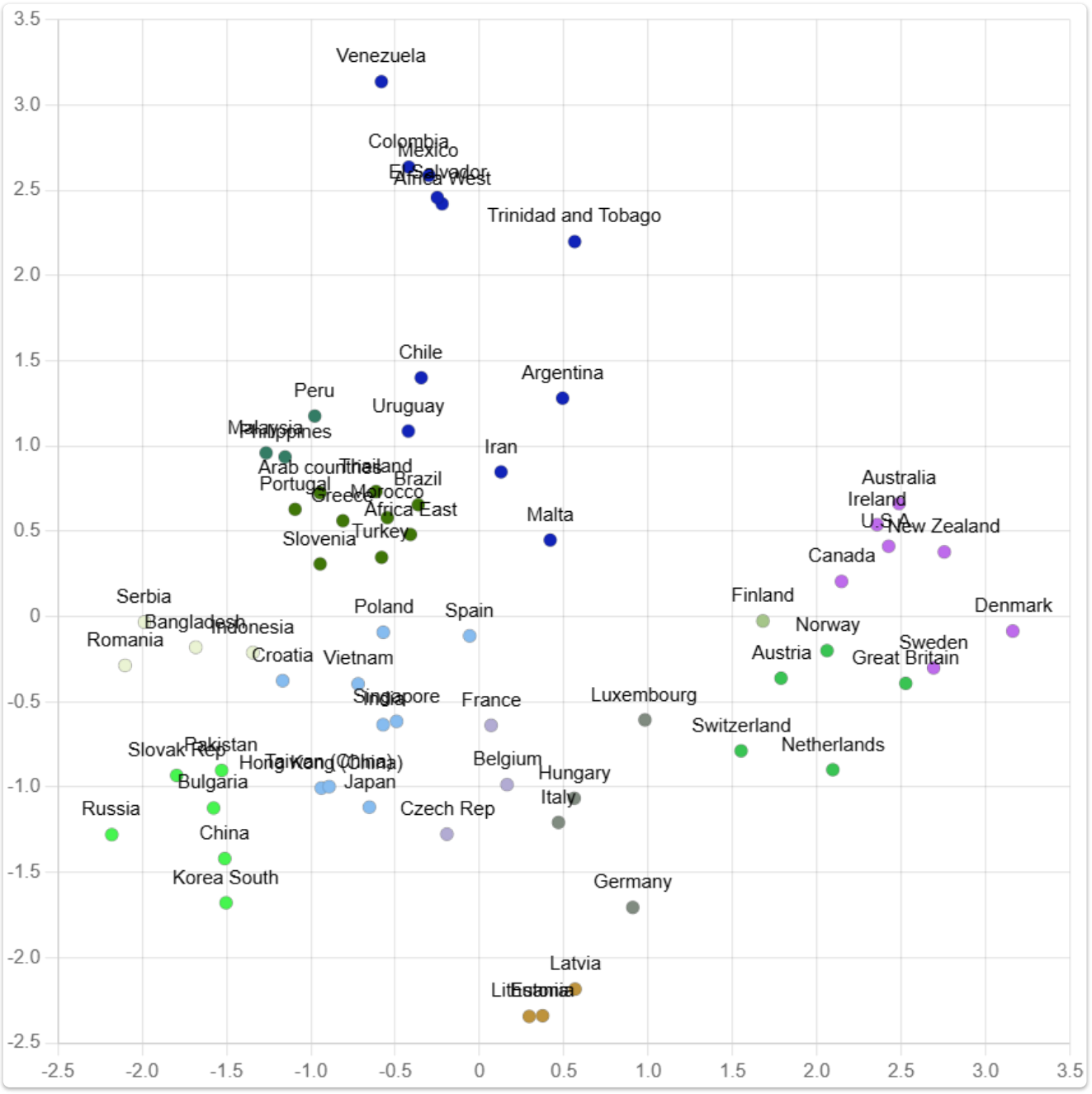
通过 PCA 降维，将六个文化维度降至二维空间。第一主成分和第二主成分的贡献率分别为 0.297 和 0.236，累计贡献率为 0.533，表明降维后的数据能够较好地保留原始数据的信息。

KMeans 聚类

选择 K=16 时 KMeans 聚类将各国文化分为 16 个簇。通过分析每个簇的中心点，我们可以总结出每个簇的文化特征。

可视化结果

在二维散点图中，每个点代表一个国家的文化，不同颜色的点代表不同的簇。如下图所示：



通过观察散点图，我们可以发现某些国家在文化维度上具有显著的相似性。其中一些是比较符合常识的，例如波罗的海三国在图中的位置十分接近，英语国家大多在同一个簇中，中日韩和南欧、西欧国家之间也比较接近。

总结和反思

使用 Hofstede 文化维度结合 PCA 降维算法和 KMeans 聚类算法能够有效分析不同国家之间的文化相似度，但这个实验仍有许多不完善的地方，例如很多国家缺少一个或多个维度的数据，并且每个国家的文化维度在学界并没有统一的标准。