University of Verona

Master's Degree in Artificial Intelligence

Project Report: Geographic Classification of F1 Circuits Using Supervised Learning

Author: Islam Taouri [VR533905]

Course: Machine Learning Deep Learning
Academic Year: 2025/2026
Date: February 17, 2026

Contents

1 Motivation and Rationale **2**

2 State of the Art and Theoretical Background **2**

2.1 K-Nearest Neighbors (KNN) 2

2.2 Support Vector Machines (SVM) 3

2.3 Dimensionality Reduction (PCA) 3

3 Objectives **3**

4 Methodology **4**

4.1 Dataset Description 4

4.2 Feature Engineering 4

4.3 Data Pre-processing 5

4.4 Implementation Details 5

5 Experiments and Results **5**

5.1 Model Configurations 6

5.2 Quantitative Results 6

5.3 Discussion and Error Analysis 6

5.3.1 Why did KNN win? 6

5.3.2 Why did Trees fail? 7

5.3.3 Failure Cases 7

6 Conclusions **7**

6.1 Future Work 8

List of Tables

1 Mean Accuracy and Standard Deviation (5-Fold CV) 6

1 Motivation and Rationale

Formula 1 is a global sport. The calendar travels to five continents, from the streets of Monaco to the deserts of the Middle East and the parks of Australia. Every circuit is unique, but they all share specific geographic properties—Latitude, Longitude, and Altitude—that define their physical location on Earth.

The main idea behind this project was simple: Can we teach a machine to figure out which continent a race track belongs to, just by looking at its coordinates?

I chose this topic for a few reasons. First, I wanted a dataset that was "real" but manageable. The F1 dataset is public and clean, which means I didn't have to spend weeks fixing missing values. Second, the problem of *Geographic Classification* is actually quite relevant in the real world. For example, in logistics or biology, you often have to group data points based on their location. Using F1 tracks is a fun way to test these algorithms in a controlled environment.

However, there was a catch. The dataset is tiny. There are only about 77 usable circuits in the history of the sport that fit my criteria. In modern Machine Learning, we usually talk about "Big Data" with millions of rows. Working with "Small Data" is actually harder in some ways because the risk of overfitting is huge. If the model just memorises the 77 coordinates, it's useless. So, a big part of my motivation was to learn how to handle small datasets properly using techniques like Cross-Validation and dimensionality reduction.

The project explores three main families of models: Distance-based (KNN), Margin-based (SVM), and Tree-based (Random Forest). I wanted to see which logic works best for geography.

2 State of the Art and Theoretical Background

Before diving into the code, I researched how other people handle spatial data classification. It turns out there are a few standard ways to do it.

2.1 K-Nearest Neighbors (KNN)

KNN is probably the most intuitive algorithm for this. It follows "Tobler's First Law of Geography," which basically says that "everything is related to everything else, but near things are more related than distant things."

In KNN, we don't really "train" a model. We just store the data. When we get a new point (a new circuit), we look at its K closest neighbors. If 3 out of 5 neighbors are in

Europe, we guess the new point is in Europe too.

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{1}$$

I used the Euclidean distance (formula above) to measure this. The tricky part is choosing K. If K is too small (like 1), the model is very sensitive to noise. If K is too big, it might smooth out the boundaries too much.

2.2 Support Vector Machines (SVM)

SVM takes a different approach. Instead of looking at neighbors, it tries to draw a line (or a hyperplane in higher dimensions) that separates the classes with the widest possible margin. For this project, I looked at two types of SVM:

- **Linear SVM:** Draws a straight line. This is fast and simple.
- **Polynomial SVM:** Can draw curved lines. This might be better because continents aren't perfect squares; their borders are curved.

2.3 Dimensionality Reduction (PCA)

Since I am working with a very small dataset, having too many features can be bad. It's called the "Curse of Dimensionality." I used Principal Component Analysis (PCA). PCA rotates the data to find the directions (components) where the data varies the most. It helps us compress the information. For example, Latitude and "Absolute Latitude" are highly correlated. PCA combines them so the model doesn't get confused by repeated info.

3 Objectives

The goal of this project isn't just to get high accuracy (which is easy on a small dataset), but to build a robust pipeline.

My specific objectives were:

1. **Data Cleaning:** Convert the raw F1 data into something a computer can understand. This involves mapping countries to regions.
2. **Feature Engineering:** Create new features that might help the model.
3. **Preprocessing:** Apply Standard Scaling (crucial for KNN) and PCA.
4. **Model Comparison:** Compare KNN, SVM, and Random Forest.

5. **Evaluation:** Use Stratified Cross-Validation to ensure the results are real and not just luck.

I also wanted to answer a specific question: *Does adding complexity (like Polynomial kernels or Random Forests) actually help when you have such small data, or is simple better?*

4 Methodology

This section explains exactly what I did, step by step.

4.1 Dataset Description

I used the file `circuits.csv`. It contains 77 rows after I filtered it. The columns I cared about were:

- `lat`: Latitude (Float)
- `lng`: Longitude (Float)
- `alt`: Altitude (Integer, in meters)
- `country`: The country where the track is.

The first problem I faced was that "Country" is not a good target variable because there are too many countries with only 1 circuit (like Azerbaijan or Sweden). You can't train a model on 1 example. So, I grouped them into **Regions**:

- **Europe:** (e.g., UK, Italy, Spain, Russia)
- **Americas:** (e.g., USA, Brazil, Canada)
- **Asia/Oceania:** (e.g., Japan, Australia, Bahrain)

I had to drop Africa because there were only 2 circuits (Kyalami and Morocco). It was too small to form a class.

4.2 Feature Engineering

I didn't just want to use the raw coordinates. I thought about what else distinguishes these regions.

- **name_length:** I counted how many letters are in the circuit's name. My hypothesis was that maybe European tracks use shorter naming conventions compared to new tracks in Asia. It was a bit of a guess, to be honest.

- **abs_lat:** I calculated the absolute value of the latitude. This tells us how far the track is from the Equator. This is useful because it groups Northern Europe together and distinguishes it from tropical places like Singapore or Brazil.

4.3 Data Pre-processing

This was the most important step. First, I used `StandardScaler`. This is mandatory for KNN and SVM. If you don't scale, the model will think Altitude (which goes up to 1000) is way more important than Latitude (which is usually under 50). Scaling forces everything to have a mean of 0 and a variance of 1.

Then, I applied PCA (Principal Component Analysis). I set the parameter to keep 95% of the variance.

- **Before PCA:** 5 features (lat, lng, alt, name_length, abs_lat).
- **After PCA:** It usually reduced down to 3 or 4 components.

This confirmed that some of my features were redundant (correlated), so PCA did its job.

4.4 Implementation Details

I wrote the code in Python using Google Colab. I used the `scikit-learn` library for everything. Here is a snippet of how I set up the pipeline:

```
# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying PCA to keep 95% variance
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

print(f"Original features: {X_scaled.shape[1]}")
print(f"Reduced features: {X_pca.shape[1]}")
```

Listing 1: Preprocessing and PCA Setup

I also used `StratifiedKFold` for validation. This is super important. Because I have different numbers of circuits per region (Europe has way more than Americas), a normal split might accidentally put all American tracks in the test set. Stratified splitting ensures the ratio stays the same.

5 Experiments and Results

I ran the experiments multiple times to make sure the results weren't just a fluke. I compared 7 different model configurations.

5.1 Model Configurations

1. **KNN (K=3):** Small neighborhood.
2. **KNN (K=5):** Balanced.
3. **KNN (K=7):** Large neighborhood.
4. **SVM (Linear):** Simple line separation.
5. **SVM (Polynomial):** Degree 3 curves.
6. **Random Forest:** 100 trees.
7. **Decision Tree:** Max depth 5.

5.2 Quantitative Results

The table below shows the accuracy results from the 5-Fold Cross-Validation.

Table 1: Mean Accuracy and Standard Deviation (5-Fold CV)

Model	Mean Accuracy	Std. Dev
KNN (K=3)	0.935	0.052
KNN (K=5)	**0.958**	**0.041**
KNN (K=7)	0.912	0.063
SVM (Linear)	0.925	0.048
SVM (Poly)	0.890	0.075
Random Forest	0.875	0.090
Decision Tree	0.810	0.112

As you can see, **KNN with K=5** was the winner. It had the highest accuracy and the lowest standard deviation, which means it was the most stable.

5.3 Discussion and Error Analysis

The results were actually quite interesting. I expected the more complex models like Random Forest to do better, but they performed worse.

5.3.1 Why did KNN win?

It makes sense if you think about it. Geography is literally about distance. The KNN algorithm is designed to calculate distances. Circuits in Europe are physically close to each other. So, using distance as the main classifier is the most logical approach.

5.3.2 Why did Trees fail?

Decision Trees and Random Forests try to draw orthogonal (straight) cuts in the data. Imagine trying to separate Europe from Asia using only horizontal and vertical lines. It's really hard. The borders are curved and diagonal. Trees struggled to capture this geometry. Also, with only 77 data points, the trees likely overfitted—they memorized the training data and couldn't generalize to the test data.

5.3.3 Failure Cases

I looked at the specific circuits that the model got wrong (the confusion matrix).

- **Turkey & Azerbaijan:** These were the biggest headaches. Geographically, they are on the border of Europe and Asia. The dataset labels them as "Europe," but their coordinates are very close to the "Asia" cluster. The model often predicted them as Asia.
- **Russia (Sochi):** Similarly, Sochi is quite far east. The model was sometimes unsure if it was Europe or Asia.
- **The Middle East:** Tracks like Bahrain and Qatar are technically Asia, but they are physically somewhat close to Southern Europe or Africa (which was removed).

It turns out that the "political" definition of continents doesn't always match the "mathematical" definition of clusters in PCA space.

6 Conclusions

This project was a great learning experience. I successfully built a pipeline to classify F1 circuits by region.

The main takeaways are:

1. Simpler is often better: On small datasets, a simple algorithm like KNN (K=5) outperformed complex ones like Random Forest.
2. Data preparation is key: Creating the "Region" target and scaling the features was more important than tuning the hyperparameters.
3. Geography is messy: Boundaries between continents aren't clear-cut lines, and models struggle with edge cases like Turkey or Russia.

To be honest, the "name_length" feature I engineered didn't help much. I analized the feature importance, and it was mostly noise. The coordinates (lat/lng) did 90% of the work.

6.1 Future Work

If I had more time, I would try to expand the dataset. 77 samples is just too small. I could include Moto GP tracks or other racing series to get the dataset up to maybe 200 or 300 samples. This would allow me to try Neural Networks, which usually fail on small data.

I would also try "Soft Classification." Instead of saying "This is definitely Europe," the model could say "This is 60% Europe and 40% Asia," which would handle the border countries much better.