Weather Prediction

Introduction

What Is Weather Prediction

Weather prediction is the application of science and technology to forecast the conditions of the atmosphere for a given location.

There are several tools and methods that are used to make the weather predictions,
Such as computer models**,** satellite imagery, and ground-based observations. These tools help meteorologists and other weather experts to understand the current state of the atmosphere and make forecasts about what the weather will be like in the future.

This research talk about Weather Prediction using AI Models:
Computer models are mathematical models that use the observed data to predict weather by considering factors such as Earth's rotation, air, water movement, and heat exchange. Meteorologists use these models to improve forecast accuracy by comparing results to actual weather patterns.

What Is The Weather Prediction Purpose

The need for weather prediction has been a topic of interest for centuries as it significantly impacts us in various areas, such as Agriculture, Aviation , Daily Decisions and alot more which get affected according to the weather prediction.

Agriculture: farmers rely on forecasts to determine when to plant and harvest their crops, as well as to plan for irrigation and pest control.

Aviation: pilots and airlines use weather forecasts to plan routes and ensure the safety of their passengers and crew.

Daily Decisions: things like what to wear or whether to have an outdoor or indoor activity, can be influenced by the weather.

By knowing the upcoming weather, we can adjust and prepare for it.

The Motivation For Selecting The Weather Prediction Topic

Weather prediction is a topic that is relevant to everyone, as the weather has a direct impact on our daily lives as I mentioned in the previous section “What Is The Weather Prediction Purpose“. We all rely on weather forecasts to make informed decisions.

As the climate crisis worsens, accurate weather prediction becomes increasingly important. Extreme weather events, such as hurricanes, heatwaves, and floods, are expected to increase in frequency and severity, making it an interesting and important topic for us to study and acknowledge. Studying this topic could also help us contribute to finding solutions in the upcoming future.

It's also a popular and rapidly evolving topic.
I know that there is a vast amount of reliable data available for this field, including AI models, code libraries, datasets, scientific articles, and much more which would help me become "experts" in the field and provide high-quality research.

The Research Materials & Models

In this research, I explore two popular models for weather prediction in the AI Field:

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Bonus: KNN-Naive Bayes Fusion

I have chosen the KNN and Naive Bayes models because of their simplicity, fast training, performance on small datasets, and ability to handle categorical features, as you will see in the following research.

Keep in mind that it's not always the case that one machine learning model fits all problems, it's crucial to take into account the unique attributes, data and the desired prediction task when deciding which model.

K-Nearest Neighbors (KNN)

KNN Explanation

K-nearest neighbors (KNN) is a simple, non-parametric machine learning algorithm used for both classification and regression.

Given a set of data points, KNN can be used to predict the value of a new data point based on its similarity to a set of existing data points. The similarity between data points is determined using a distance metric, which is a function that calculates the distance between two points in space.
The metric distance measured by Manhattan Distance, Minkowski Distance and others.
With the most common one which is:

Euclidean Distance $d(x,\ y)\ =\ \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$

Note: Each distance measurement has its own advantages.

In classification tasks, the dependent variable is categorical and the goal is to assign a new data point to one of the available categories.
To do this, KNN identifies the K data points in the training set that are most similar to the new data point based on the chosen distance metric. It then counts the number of data points in each category among these K nearest neighbors and assigns the new data point to the category where the majority of the K nearest neighbors belong.

In regression tasks, the dependent variable is continuous and the goal is to predict the value of a new data point based on its similarity to a set of existing data points.

To make a prediction, KNN identifies the K data points in the training set that are most similar to the new data point based on the chosen distance metric. It then takes the mean or median value of the dependent variable among these K nearest neighbors and uses this value as the prediction for the new data point.

For an in depth of the KNN model check the following link:
[The KNN Model - Springer]

KNN Motivation

KNN is a straightforward and intuitive learning algorithm that is commonly used for classification tasks,
it is easy to understand and implement, making it a good choice for students which have limited background in machine learning.

Its ability to handle both continuous and categorical data, making it a versatile choice for weather prediction tasks that involve a mix of different data types.

- Continuous data: Temperature, Precipitation, Humidity, Wind speed, etc.
- Categorical data: Cloud cover and weather conditions such as Rain, Snow, etc.

As I explained earlier, the KNN model has the ability to make predictions based on the similarity of neighboring data points, which can be particularly useful in weather prediction tasks where the behavior of nearby weather patterns may have an influence on future conditions.

KNN has the ability to make predictions without the need for extensive training data, which can be a useful feature in situations where data is limited or difficult to obtain.

Lastly, KNN does not make any assumptions about the underlying data distribution (non parametric), which makes it a robust method for weather prediction.

Finding The Optimal KNN Variant: Uniform Vs Weighted

In K-Nearest Neighbors (KNN), the value of K determines the number of nearest neighbors used to make predictions. Choosing the optimal value of K is crucial for the performance of the KNN model.

In this study, I compared the performance of the KNN model using two variants: Uniform & Weighted.
I tested a range of K values from 1 to 50 and divided the data into Training set and Test set.
I trained the KNN model on the training set using different values of K and then evaluated the model's performance on the test set.

The weighted variant I used had a weighting factor in the form of 1/d,
where d is the distance from the selected neighbors to the point being examined.
This means that the closer a neighbor is, the more influence it has on the prediction.
In contrast, in the uniform model, each neighbor has an equal influence, or voting power, of 1.

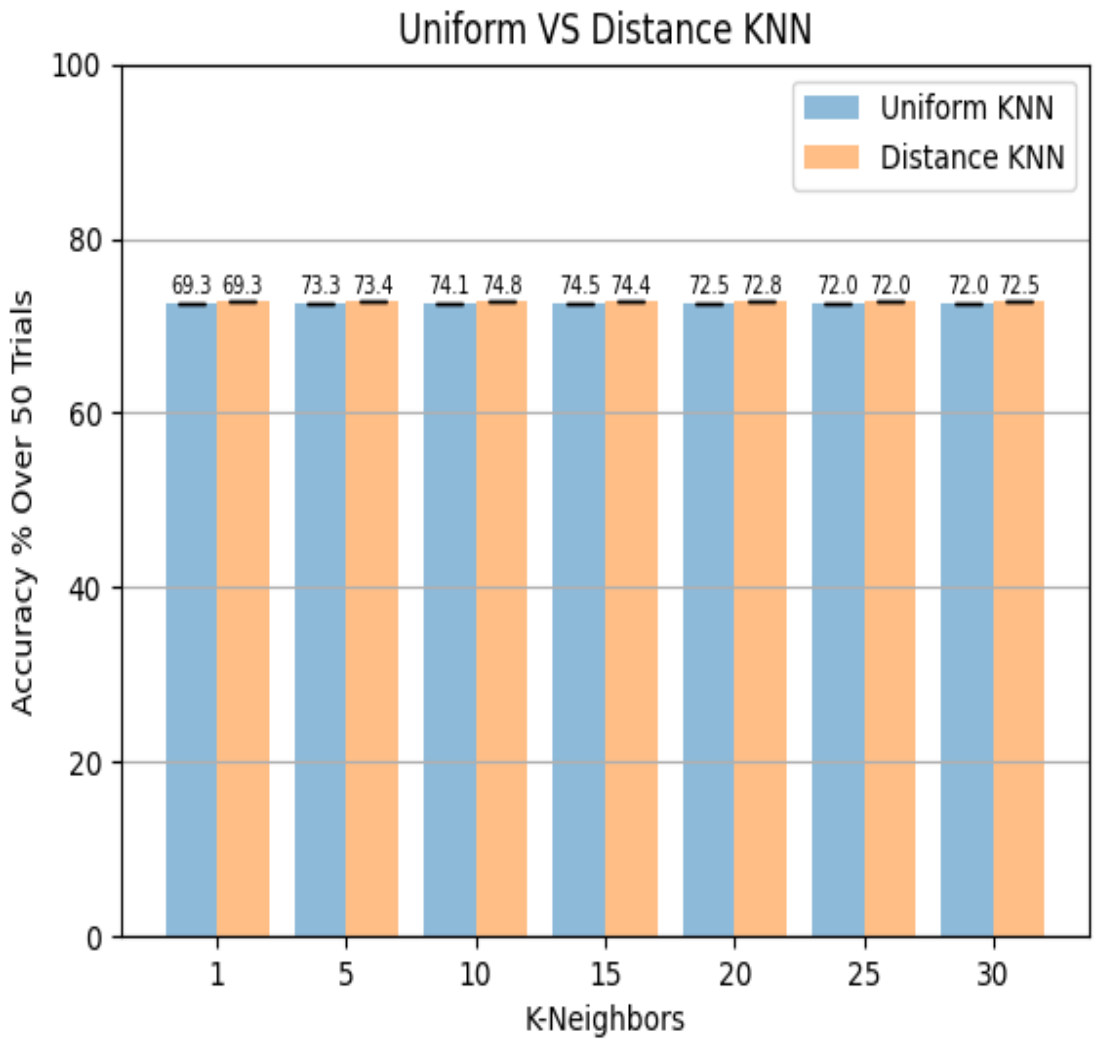
I found that the results of both variants were close with the weighted variant having a slightly better performance. The weighted variant peaked at 74.8% on K=10,
while the uniform variant peaked at 74.05% on K=15.

Additionally, I observed that as K increased, the uniform variant's performance dropped while the weighted variant maintained a high level of accuracy.
This indicates that the weighted variant is more robust and can maintain a high level of accuracy across a wider range of K values while the uniform variant is more sensitive to the choice of K.

It's worth noting that the outcome may vary depending on the chosen dataset. In this particular case, the dataset used contains approximately 2800 records, which is relatively low. As a result, the accuracy results were fairly equal.
However, if the dataset were larger, it's likely that the weighted variant would have significantly outperformed the uniform variant, particularly as the value of K increases.

Overall, our results suggest that using a weighted approach can be beneficial for improving the accuracy and robustness of the KNN model.

Note: From now on, when I refer to the KNN variant, I will refer to Weighted K = 10.



Code link for creating the graph: KNN Variants Comparison
Dataset link for the models: Database

Model Overview (Implemented Code)

Code Link: KNN

1. Importing the necessary libraries:
   The first line of code imports the pandas library, which is used for data manipulation and analysis.
   The second line imports the accuracy_score function from the sklearn.metrics library, which is used to evaluate the performance of the model by comparing the predicted output and actual output.
   The third line imports the train_test_split function from the sklearn.model_selection library, which is used to split the data into a training set and a test set.
   The fourth line imports the KNeighborsClassifier class from the sklearn.neighbors library, which is the implementation of the KNN algorithm in scikit-learn.

2. Loading the data:
   The code uses the read_csv() function from pandas to load the data from a CSV file called "weather.csv" and store it in a variable called "data".

3. Extracting the input features and output column:
   The code uses the drop() function from pandas to remove unneeded columns from the data, which are "location", "date" and “weather”, and store the remaining columns in the variable "x_input".
   The code also uses the same function to extract the "weather" column from the data and store it in the variable "y_output".

4. Splitting the data:
   Using the train_test_split function from sklearn.model_selection, the data is splitted into a training set and a test set.
   The input features and output column are passed as the first and second arguments, respectively.
   The test_size argument is set to 0.2, which means that 20% of the data will be used for testing and the rest for training.

5. Creating the model:
   The code creates an instance of the KNeighborsClassifier class and sets the number of nearest neighbors to 10 and defines the weight method as distance (We determined that weighted KNN is better for us for weather prediction in the section above).

6. Training the model:
   The code uses the fit() method from the KNeighborsClassifier class to train the model on the training data.
   The x_train and y_train variables, which contain the input features and output column of the training data, are passed as arguments.

7. Making predictions:
   The code uses the predict() method from the KNeighborsClassifier class to make predictions on the test data using the trained model.
   The x_test variable, which contains the input features of the test data, is passed as an argument. The predictions are stored in the variable "y_weighted_prediction".

8. Computing the accuracy:
   The code uses the accuracy_score() function from the sklearn.metrics library to compute the accuracy of the model's predictions on the test data.
   The function takes in the true values in the test set and predicted values as its arguments.
   The output of the function is a decimal value between 0 and 1, which represents the proportion of correct predictions made by the model.
   The code then rounds it off to four decimal places and converts it to be formatted in percentage view and then print the accuracy.

Naive Bayes

Naive Bayes Explanation

Naive Bayes is a probabilistic machine learning algorithm that is used for classification tasks.
It is based on the Bayes theorem, which states that the probability of an event occurring is equal to the prior probability of the event multiplied by the likelihood of the event given the evidence.
In the context of classification, the goal is to predict the class label of a given data point based on a set of features.

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

which is equivalent to saying the following:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

By calculating the prior of a class from the training set, and calculating the mean and variance of each feature class. We could find
$p(x = v \mid C_k)$

by using the normal distribution (Gaussian naive Bayes) as follows:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

where Uk is the mean of values x in the class Ck and σ^2 is the variance of the values x in class Ck.

The Naive Bayes algorithm makes the assumption that all of the features are independent of each other, hence the name "naive."
This assumption is often unrealistic, but the algorithm still performs well in practice.

To classify a new data point, the Naive Bayes algorithm calculates the probability of the data point belonging to each class, based on the class's prior probability and the likelihood of the features given the class.
The class with the highest probability is chosen as the prediction for the new data point.

For an in depth of the Naive Bayes model check the following link:
Academic - Naive Bayes - Springer
Intuitive - Naive Bayes - Python Sklearn

Naive Bayes Motivation

Naive Bayes is a fast and simple algorithm that can be trained on a small dataset and still produce reliable results. Particularly useful for weather prediction, where the data is often noisy and there may not be a large amount of historical data available.

Naive Bayes can handle multiple features and can be used for both binary and multi-class classification tasks.
In weather prediction, there may be multiple features that are relevant, such as Temperature, Humidity, Wind speed, Pressure, and Naive Bayes can effectively handle all of these features simultaneously.

Lastly, Naive Bayes is known to be a robust algorithm that performs well even when the underlying assumptions of the model are not completely met.
This is important in the case of weather prediction, as the conditions and patterns of the atmosphere are complex and may not always behave in a predictable manner.

Finding The Optimal Gaussian Naive Bayes: Prior of 1/K VS Estimate Prior

In the Naive Bayes approach I consider two ways of choosing the class prior:

1. Dividing by the total number of classes: (Prior of 1/K)
   This method involves setting the class prior to be equal to 1 divided by the total number of classes in the dataset.
   This approach assumes that all classes are equally likely to occur and therefore assigns equal weight to each class.

2. Dividing the number of occurrences of each class in the training data by the length of the training data: (Estimate Prior)
   This method involves calculating the class prior based on the relative frequency of each class in the training data.
   For example, if a particular class occurs in 20% of the training data, the class prior for that class would be set to 0.2.
   This approach allows the model to take into account the relative frequencies of different classes in the training data, which may be more accurate in cases where certain classes are more likely to occur than others.

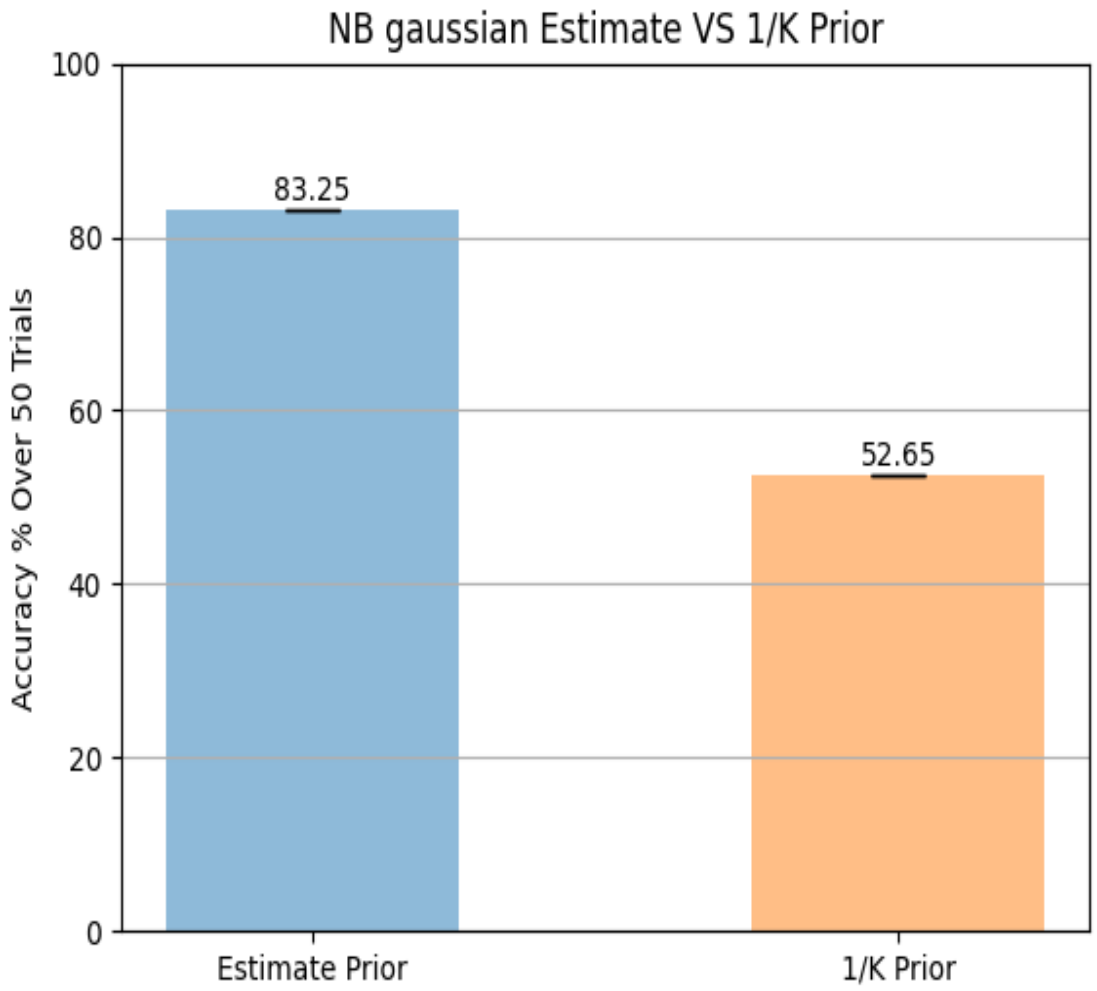
I chose the second method (Estimate Prior) because it allows the model to better capture the underlying patterns in the data.

In our experiment, the second method proved superior with an average accuracy of 83.25% while the first method had an accuracy of 52.65%.

It is important to note that the choice of class prior can have a significant impact on the performance of a Naive Bayes model.

Therefore, it is essential to consider which approach is most appropriate for a given dataset.

Note: From now on when I talk about Gaussian NB i will refer to Estimate Prior (second method).



Code link for creating the graph: [Gaussian NB Prior Comparison]
Dataset link for the models: [Database]

Model Overview (Implemented Code)

Code Link: Gaussian NB

1. Importing the necessary libraries:
   The first line of code imports the pandas library, which is used for data manipulation and analysis.
   The second line imports the accuracy_score function from the sklearn.metrics library, which is used to evaluate the performance of the model by comparing the predicted output and actual output.
   The third line imports the train_test_split function from the sklearn.model_selection library, which is used to split the data into a training set and a test set.
   The fourth line imports the GaussianNB class from the sklearn.naive_bayes library, which is the implementation of the Gaussian Naive Bayes algorithm in scikit-learn.

2. Loading the data:
   The code uses the read_csv() function from pandas to load the data from a CSV file called "weather.csv" and store it in a variable called "data".

3. Extracting the input features and output column:
   The code uses the drop() function from pandas to remove unneeded columns from the data, which are "location", "date" and “weather”, and store the remaining columns in the variable "x_input".
   The code also uses the same function to extract the "weather" column from the data and store it in the variable "y_output".

4. Splitting the data:
   Using the train_test_split function from sklearn.model_selection, the data is splitted into a training set and a test set.
   The input features and output column are passed as the first and second arguments, respectively.
   The test_size argument is set to 0.2, which means that 20% of the data will be used for testing and the rest for training.

5. Creating the model:
   The code creates an instance of the GaussianNB class and it doesn't require any specific parameter (By not passing a value of Priors, the created instance is Estimate Prior Gaussian NB, the choice to use Estimate Prior Gaussian NB was made as a result of the experiment from the previous section).

6. Training the model:
   The code uses the fit() method from the GaussianNB class to train the model on the training data.
   The x_train and y_train variables, which contain the input features and output column of the training data, are passed as arguments.

7. Making predictions:
   The code uses the predict() method from the GaussianNB class to make predictions on the test data using the trained model.
   The x_test variable, which contains the input features of the test data, is passed as an argument. The predictions are stored in the variable "y_gaussian_prediction".

8. Computing the accuracy:
   The code uses the accuracy_score() function from the sklearn.metrics library to compute the accuracy of the model's predictions on the test data.
   The function takes in the true values in the test set and predicted values as its arguments.
   The output of the function is a decimal value between 0 and 1, which represents the proportion of correct predictions made by the model.
   The code then rounds it off to four decimal places and converts it to be formatted in percentage view and then print the accuracy.

Data & Training Model

The Dataset

In the research I used a dataset which has approximately 2800 data points.
The dataset is organized in excel file with the following columns:

- location: string, Representing the city that the data took (New-York / Seattle).
- date: YYYY-MM-DD, representing the date the data took.
- precipitation: Unsigned float, representing the precipitation amount.
- temp_max: Float, representing the maximum temperature.
- temp_min: Float, representing the minimum temperature.
- wind: Unsigned float, representing the wind speed.
- weather (output): Limited list, representing the weather status.

The date and location columns may introduce noise into the dataset which can make it difficult for the model to identify important features related to weather prediction, leading to poor performance and it could make the dataset larger and more complex, increasing computation time and memory required to train the model.
Therefor I decided to remove those columns from the dataset

The weather is calculated based on the columns: Precipitation, Temp_max, Temp_min, and Wind.
The weather classification has one of the following values: Drizzle, Rain, Sun, Snow and Fog.

Link to the dataset: Database

Test Data Splits

The data in the experiments was split into Training and Test sets with a ratio of 0.8,
with the data being randomly allocated to each set.
Means that approximately 80% of the data was used for training and the remaining 20%
was used for testing the model accuracy.

The use of a randomly determined training and test split has the advantage of ensuring that the results are not biased by any particular ordering of the data.
However, it also means that the results may vary due to the randomness of the split.

Models Comparison: KNN & Naive Bayes

Reminder:

KNN - refer to Weighted KNN (K = 10), I chose the superior one, check KNN comparison.
NB - refer to Estimate Prior Gaussian NB, I chose the superior, check NB comparison.

Similarity

- Supervised Learning Algorithms:
  Both k-nearest neighbors (KNN) and Naive Bayes are supervised learning algorithms.
  This means that they both require labeled training data for their operation.
  The training data is labeled with the correct output or class, and the algorithm learns to make predictions on new unseen data based on this training data.

- Concept of Similarity:
  Both KNN and Naive Bayes models are based on the concept of "similarity".
  In KNN, similarity is defined by the distance between instances in the feature space, while in Naive Bayes, similarity is defined by the conditional probability of each class given the input features.

- Simplicity and Implementability:
  Both KNN and Naive Bayes are relatively simple to understand and implement.
  The concepts behind the algorithms are easy to grasp and they can be implemented using only a few lines of code.

- Computational Efficiency:
  Both KNN and Naive Bayes are computationally efficient algorithms that don’t require a lot of computational resources to train.

- Lazy Learning:
  Both KNN and Naive Bayes are "lazy learners”, which means that they don't explicitly learn a model for the underlying data distribution.
  Instead, KNN stores the training instances and makes predictions based on the stored instances, while Naive Bayes uses the training instances to estimate probability densities and makes predictions based on these densities.

Differences

- Decision boundary:
  KNN generates a non-linear decision boundary and is based on the majority class or average of the k-nearest data points in the feature space, while Naive Bayes generates a linear decision boundary and is based on the probability of each class given the input features.

- Assumptions:
  KNN makes no assumptions about the underlying distribution of the data, while Naive Bayes makes the "naive" assumption that all features are independent of each other, which is rarely the case in real-world data.

- Computational complexity:
  KNN has a high computational complexity during the testing phase and requires large amounts of storage for storing the training instances, while Naive Bayes has a low computational complexity and requires low memory during testing.

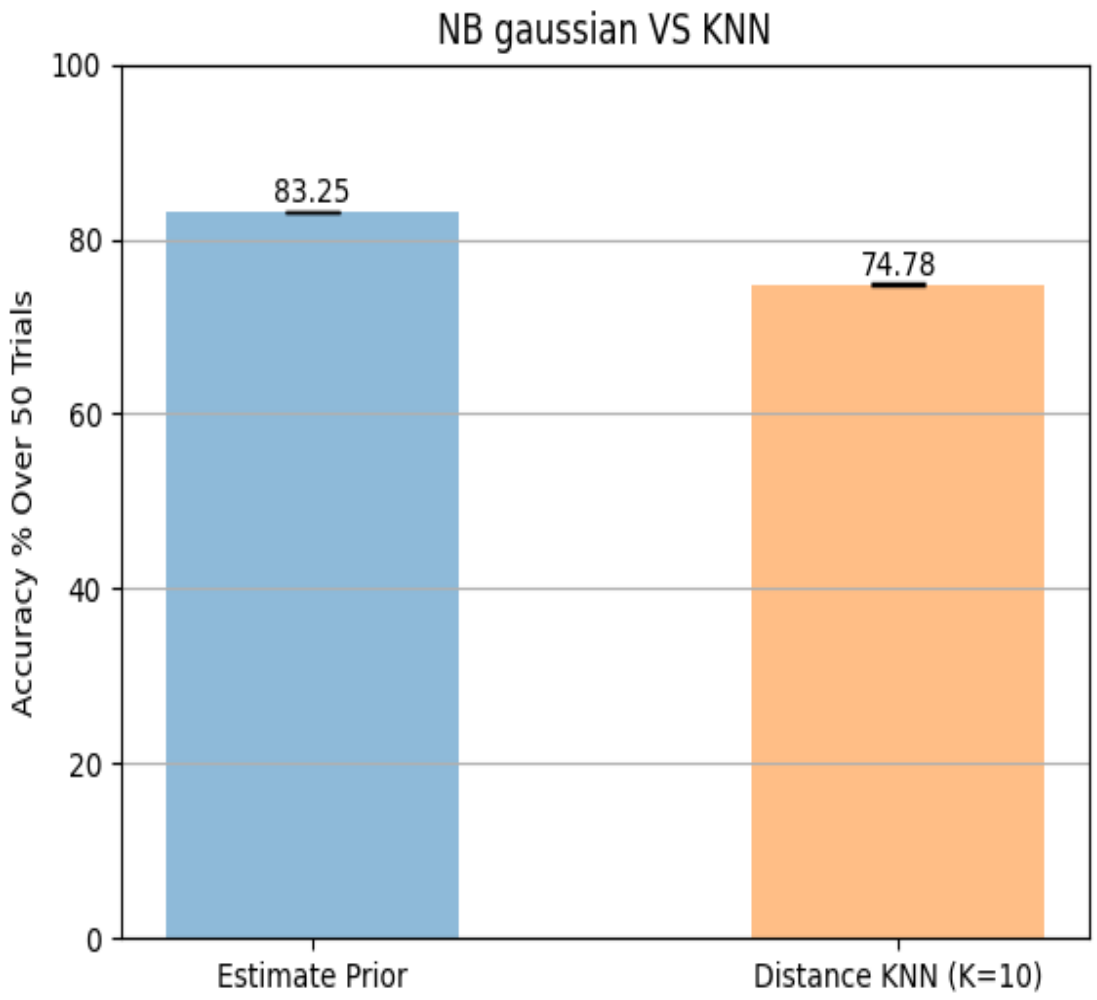
- Robustness:
  KNN is sensitive to noisy data and the choice of the hyperparameter k, while Naive Bayes is less sensitive to noisy data and the choice of parameters.

Performances

In comparing the effectiveness of K-Nearest Neighbors (KNN) and Naive Bayes (NB) for weather prediction, it appears that Naive Bayes outperforms the KNN.
The average accuracy of Weighted KNN using K = 10 was 74.78% over 50 trials, while the accuracy of Estimate Prior Gaussian Naive Bayes was 83.25%

These results suggest that Naive Bayes may be a particularly effective choice for weather prediction in this context, although a larger dataset and a wider range of weather conditions would be needed to confirm or refute this conclusion.

One possible reason for the better performance of Naive Bayes is its robustness to irrelevant features.
In the context of weather prediction, this means that a Naive Bayes classifier may be more accurate even if it is given features that are not directly related to the weather, because it can ignore these features.
On the other hand, KNN uses all features equally when making predictions, so it may be more affected by noise from irrelevant features.



Code link for creating the graph: [KNN & NB Comparison]
Dataset link for the models: [Database]

KNN-Naive Bayes Fusion: A Hybrid Approach for Feature Probability Estimation

Introduction

In this research, I propose a novel hybrid model that combines the strengths of K-Nearest Neighbors (KNN) and Naive Bayes (NB) for improved feature probability estimation in weather prediction.

The unique aspect of our approach is the use of KNN to find the K nearest neighbors of a given input, and then using Naive Bayes to calculate the probability of each feature based on the feature values of the K nearest neighbors.

Background

One of the key advantages of this approach is that it allows us to take into account the local feature-class relationships, while still accounting for feature independence. It is particularly useful in cases where the data is noisy or has a lot of outliers, as the K nearest neighbors can help to smooth out these irregularities and improve the accuracy of the predictions made by the Naive Bayes model.

Additionally, this approach can help to handle non-linear decision boundaries by using the K nearest neighbors to estimate the decision boundary, making the model more robust to non-linear relationships between the features and the class labels.

Methods

My approach starts by using KNN to find the K nearest neighbors of a given input, and then using Naive Bayes to calculate the probability of each feature based on the feature values of the K nearest neighbors. I have implemented this technique using python and tested it using a weather dataset.

To test the performance of our model, I have compared it to the Gaussian Naive Bayes model estimated prior.

Results

The experimental results show that the KNN-Naive Bayes model performs better than the Gaussian Naive Bayes model estimated prior, given that K is bigger than 1% of the training size. The normal Naive Bayes model achieved an accuracy of 85.82%, while the KNN-Naive Bayes model achieved an accuracy of 83.47% when only considering the 1% closest neighbors. However, as the number of closest neighbors considered increases, the accuracy of the KNN-Naive Bayes model also increases (peaking at 10%).
The model achieved an accuracy of 86.058% when considering the 5% closest neighbors, an accuracy of 86.093% when considering the 10% closest neighbors, an accuracy of 86.075% when considering the 15% closest neighbors, and an accuracy of 86.074% when considering the 20% closest neighbors. The highest relative increase being 0.32% at 10% of the training size.

Discussion and Insights

My proposed technique of combining KNN and NB for weather prediction is a creative and promising approach. It has the potential to improve the accuracy of predictions made by the Naive Bayes model, particularly in cases where the data is noisy or has a lot of outliers. However, the technique can increase the runtime of the model. To overcome this limitation, more efficient algorithms for finding nearest neighbors could be considered.
Additionally, further research could explore the use of this technique in other domains and applications.

Future Work

- Investigate more efficient algorithms for finding nearest neighbors, such as kd-trees or locality sensitive hashing.

- Test the model in different domains and applications, to see if the technique can be generalized to improve feature probability estimation in other areas.

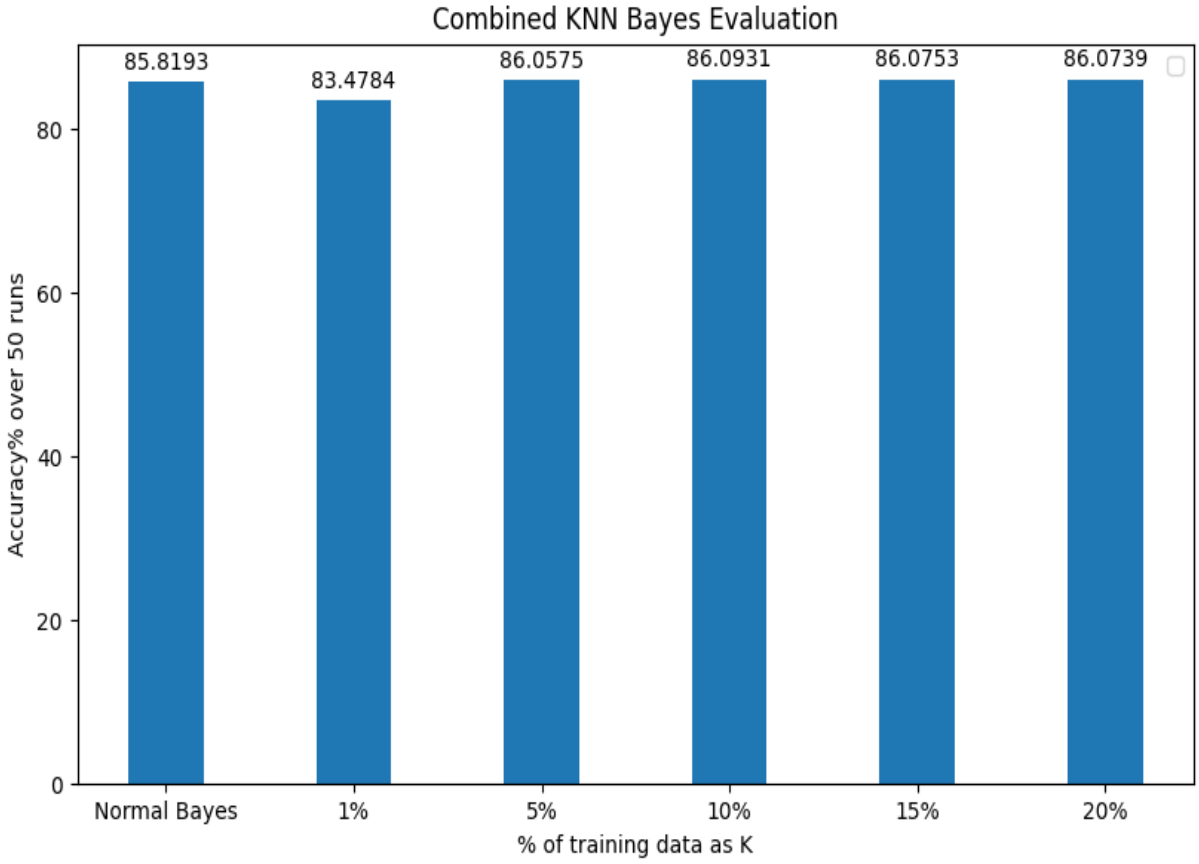
- Test the model with a bigger dataset, as a larger dataset size may have a significant impact on the performance of the model.

Limitations

- The results of this study are based on a specific dataset size of 2800 records.

- The comparison between the Gaussian Naive Bayes model estimated prior and the proposed KNN-Naive Bayes model would have shown a bigger gap in the accuracy of the KNN-Naive Bayes model with a larger dataset.

- Further research is needed to explore the impact of increasing the dataset size on the performance of the model.



Code link for creating the graph: KNN_NB_Fusion
Dataset link for the models: Database

Research Overview

Difficulties In The Research

- Lack of domain expertise :
  Weather prediction is a complex field that requires a thorough understanding of meteorological principles and processes.
  Without having prior knowledge or experience in this domain, it was difficult to fully understand the model's predictions and evaluate their accuracy.

- Time constraints :
  Conducting a thorough review of an AI model is a time-consuming process, especially if the model is complex and the data set is large.
  As a student I didn’t have enough time to fully analyze the model and its performance, which could affect the quality of the research.

- Choosing the data split :
  A big dilemma in training a machine learning model is finding the balance between using enough data to train the model effectively, while also leaving enough data for testing and evaluation.
  A large training set can allow the model to learn better and achieve improved performance, but if the model is too complex or the training set is too large, the model may overfit according to the training data, resulting in poor generalization to unseen examples.
  On the other hand, if the training set is too small, the model may not be able to learn effectively and will perform poorly.

Insights and Lessons Learned

- Expanding my understanding of the reviewed models :
  By gaining a comprehensive understanding of the K-Nearest Neighbors (KNN) and Naive Bayes models, I have been able to not only use them effectively, but also understand their underlying principles and assumptions.
  This included knowledge of when and why these models were appropriate to use, as well as how to implement them in practice.
  Additionally, I had learned how to evaluate the performance of these models and how to optimize their parameters for a given dataset.
  Overall, the goal was to become proficient in these models and understand how they can be leveraged to solve a wide range of problems.

- Feature selection and engineering :
  Careful selection and construction of the input features for the machine learning models can have a large impact on the performance of the model.

- Working with AI libraries :
  The value of being familiar with libraries like scikit-learn, pandas, numpy, and matplotlib for machine learning and data analysis, which can greatly streamline the process of developing and evaluating models and allows to focus on the problem, access a wide range of preprocessing, modeling and evaluation techniques that have been developed and tested by experts, making it less prone to errors, quickly experiment with different models and techniques, it is an essential skill for anyone working on AI projects.

- Model evaluation :
  It is important to evaluate the model performance using various metrics such as accuracy, mean square error, and cross-validation.

- Models accuracy result even with low data processing :
  Another insight that can be gained from working on this research is the realization that even small datasets can produce good results.
  In some cases, a relatively small dataset of weather data, combined with the right feature selection and model selection can lead to a model that performs well on the given task.

Links

K Nearest Neighbors

1. Model review :
   The KNN Model - Springer

2. Code (Libraries) :
   Uniform
   Weighted

3. Code (Hardcode) :
   Weighted

4. Code (Comparison Graph) :
   Comparison

Naive Bayes

1. Model review :
   Naive Bayes - Springer
   Naive Bayes - Python Sklearn
   KNN and Naive Bayes Comparison

2. Code (Libraries) :
   Estimate Prior
   1/K Prior

3. Code (Hardcode) :
   Estimate Prior

4. Code (Comparison Graph) :
   Comparison

KNN NB Fusion

5. Code (Hardcode) :
   Fusion

6. Code (Comparison Graph) :
   Comparison

Dataset

Database

Origin DataBase