Seattle Weather Prediction Using AI Algorithms
Team Project 03

Archana Deepak Kumar,
Ali Hashemi, Jianting Liu, Nicole Hessner
Team 05, DS510 – AI for Data Science, MS Computer Science,
School of Technology and Computing, City University of Seattle
deepakkumararchana@cityuniversity.edu
hashemiali@cityuniversity.edu
liujianting@cityuniversity.edu
hessnernicole@cityuniversity.edu

Abstract

This paper focuses on the Classification model of Seattle weather prediction based on the inputs such as date, min temperature, max temperature, wind, and precipitation, and predicts the weather condition as either Sunny, Rainy, Drizzling, Snowy, or Foggy for a given day. The AI algorithms such as K-Means Clustering, Decision Trees, and others will be used to predict the weather changes in Seattle. The Data set that will be used for Seattle weather prediction is from the online site Kaggle.com which contains 1400+ input records (Kaggle, 2020). Since this Kaggle dataset only goes through 2015, any model that we develop can also be compared to weather data between 2015 and 2021. If the more current data does not come from the same weather center, the model will inherently not be as accurate, so we will attempt to determine the correct weather center. If it is not able to be determined, the model should be able to serve as a close approximation and can be compared as such. The best algorithm for weather prediction will be selected based on the predictions/accuracy comparison from different algorithms that will be discussed in this paper.

Keywords: Seattle weather, K-means clustering, decision tree, temperatures, accuracy, AI (Artificial Intelligence), ML (Machine Learning)

1. INTRODUCTION

Machine learning is a subset of AI which helps a system to generate accurate output based on the training provided on existing datasets. Machine Learning algorithms look for patterns within the training dataset and learn to predict without human help. (Rothman, 2018)

As part of this paper, we will learn to predict the weather in Seattle based on the temperatures, wind, and precipitation input values. This prediction will be based on the models we train to predict the weather using machine learning algorithms. After testing several common algorithms, we will determine which one(s) perform the best for this problem. We will do this by comparing the accuracy scores of the various models when tested against a reserved segment of the dataset.

Supervised Learning is one of the traditional methods of Machine Learning in which the computer helps humans identify the patterns and relationships in the data to predict the outcome of new data. The weather prediction problem is one that typically uses supervised learning to find the pattern. (Rothman, 2018)

For the problem of weather prediction, we will be comparing Logistic Regression, Naïve-Bayes, K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Support Vector Classifier, and Stochastic Gradient descent models. Each model will utilize

70% of the data as a training dataset and 30% of the data as a testing dataset.

This paper is focused on the real-world effects and uses of AI in local weather forecasting. Not only can this data be useful for everyday planning, but it also can be used in extreme or disaster situations. Since weather is complex to interpret and is influenced locally by minor changes worldwide, the strong and efficient components of AI can be extremely useful in aiding human intelligence with this problem.

2. LOGISTIC REGRESSION

The Logistic Regression, also known as Logit model, is a classification and prediction algorithm that is used to solve binary outcome problems based on a set of independent variables. This algorithm is one of the easiest ones to train and implement when compared to other algorithms and works well when the data is linearly separable. This algorithm estimates the probability of an event occurring where the logistic function is represented using the formula in Equation 1, where logit(π) is the dependent variable (Thanda, 2022).

Equation 1: Logistic Regression equation.

$$Logit(\pi) = \frac{1}{e^{-\pi}}$$

The accuracy of the Logistic Regression model for predicting Seattle Weather using the training dataset is 85.03%, and the accuracy using the testing dataset is 83.83% as shown in Figure 2, based on the code shown in Figure 1.

```
# Calculate the accuracy of weather prediction through Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV

#Build and fit the model
logistic= LogisticRegression(max_iter=1200)
parameters = [{'penalty':['l2']},
              {'C':[1, 10]}]
grid = GridSearchCV(logistic,
                    param_grid = parameters,
                    scoring = 'accuracy',
                    cv = 5,
                    verbose=0)

grid_search = grid.fit(X_train,Y_train)
print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
grid.fit(X_train,Y_train)
#Making predictions on test data
Y_pred = grid.predict(X_test)
# calculating the accuracy of the model
LR_accuracy = round(accuracy_score(Y_test,Y_pred)*100,2)
# Plotting the confusion matrix on testing data
print(confusion_matrix(Y_test, Y_pred ))
print("Logistic Regression Model accuracy on Test dataset (in %):",LR_accuracy)
```

Figure 1: Code for Logistic Regression Classifier

```
{'C': 10}
Accuracy for our training dataset with tuning is : 85.03%
[[  0   0   0   0  10]
 [  0   0   0   0  41]
 [  0   0 182   1  11]
 [  0   0   8   3   0]
 [  0   0   0   0 183]]
Logistic Regression Model accuracy on Test dataset (in %): 83.83
```

Figure 2: Accuracies for Logistic Regression Classifier

3. NAÏVE-BAYES

The Naïve-Bayes algorithm examines a set of features and then determines how likely an outcome is based on that dataset. It does this by examining each feature independently to determine the probability of that feature influencing the outcome. Mathematically, this is stated as Equation 2 (Brownlee, 2020).

Equation 2: Bayes' Theorem

$$P(A|B) = \frac{(P(B|A) * P(A))}{P(B)}$$

Then, it looks like it combines the outcome probabilities of the whole feature set to predict the outcome based on overall conditions. This is known as the *maximum a posteriori* (MAP) hypothesis and can be written as any iteration of Equation 3 (Brownlee, 2020).

Equation 3: maximum a posteriori hypothesis

$$MAP(A) = \max(P(A|B))$$

or

$$\mathbf{MAP(A) = max \frac{(P(B|A) * P(A))}{P(B)}}$$

or

$$MAP(A) = \max(P(B|A) * P(A))$$

Our model is specifically based on Gaussian Naïve-Bayes, which assumes that the class-conditional densities (probability of an outcome based on a feature) are normally distributed (Hrouda-Rasmussen, 2022). The accuracy of our Gaussian Naïve-Bayes model for predicting Seattle weather using training data set is 84.93% and Testing dataset is 84.51%, based on the code shown in Figure 3.

```
# Calculate the accuracy of weather prediction through Naive Bayes Classifier
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV

#Build and fit the model
gaussian = GaussianNB()
param_grid = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
grid = GridSearchCV(gaussian,param_grid, cv =5)
grid_search = grid.fit(X_train,Y_train)
print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
grid.fit(X_train,Y_train)
#Making predictions on test data
Y_pred = grid.predict(X_test)
# calculating the accuracy of the model
NB_accuracy = round(accuracy_score(Y_test,Y_pred)*100,2)
# Plotting the confusion matrix on testing data
print(confusion_matrix(Y_test, Y_pred ))
print("Gaussian Naive Bayes model accuracy on Test dataset (in %):",NB_accuracy)
✓ 3.2s
{'var_smoothing': 5.336699231206313e-06}
Accuracy for our training dataset with tuning is : 84.93%
[[  0   0   0   0  10]
 [  0   0   0   0  41]
 [  0   0 179   4  11]
 [  0   0   2   9   0]
 [  0   0   0   0 183]]
Gaussian Naive Bayes model accuracy on Test dataset (in %): 84.51
```

Figure 3: Code and accuracies for Naïve Bayes Classifier

3. K-NEAREST NEIGHBORS

The K-Nearest Neighbors (KNN) algorithm, "[U]ses proximity to make classifications or predictions about the grouping of an individual data point, (IBM Integrated Analytics System, 2021)" typically for classification. Voting of the nearest neighbors to a point is used to determine which group a data point should belong to.–The size of the voting pool is a value K, which should be an odd number to prevent ties. Determining the best K value for the data set is an important part of optimizing the model (IBM Integrated Analytics System, 2021).

For our weather model, we tested values of K ranging from 1 to 10 as shown in Figure 4 and found that the best K value for this data set was 9, as shown in Figure 5. This means that the location of each test point was determined by whichever group had the most votes out of 9. The best accuracy for KNN with Training dataset was 75.33% and using Testing dataset, the accuracy was 74.26%.

```
# Making predictions and calculating accuracy using KNN classifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
#Using k = 1 to 10
k_range = range(1,11)
param_grid = dict(n_neighbors=k_range)

KNN_accuracy = []
for k in k_range:
    classifier2 = KNeighborsClassifier(n_neighbors=k)
    # defining parameter range
grid = GridSearchCV(classifier2, param_grid, cv=10, scoring='accuracy', return_train_score=False,verbose=1)
    #fit the model
grid_search = grid.fit(X_train,Y_train)
#Make predictions
print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
grid.fit(X_train,Y_train)
predictions=grid.predict(X_test)
KNN_accuracy = round(accuracy_score(Y_test,predictions)*100,2)
# Plotting the confusion matrix on traing and testing data
print(confusion_matrix(Y_test, predictions))
print("KNN classifier model accuracy on Test dataset (in %):",KNN_accuracy)
```

Figure 4: Code for KNN classifier

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
{'n_neighbors': 9}
Accuracy for our training dataset with tuning is : 75.33%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[[  3   0   1   0   6]
 [  0   2   3   0  36]
 [  0   3 159   0  32]
 [  0   0   8   1   2]
 [  0   6  16   0 161]]
KNN classifier model accuracy on Test dataset (in %): 74.26
```

Figure 5: KNN classifier accuracy for best k value

Since KNN is used for classification, it makes sense that it does not have the best performance for weather prediction, which is inherently probabilistic since there are so many complex and interrelated variables that influence the weather. KNN may perform better for a problem such as sorting weather conditions by season.

4. DECISION TREES

Decision Tree Classifiers capture "descriptive decision-making knowledge (Galarnyk, 2022)" from a dataset. The goal of a decision tree is to use a series of binary decisions using the features of the dataset to categorize the data into a set of descriptive categories. The initial feature used to make a decision about the data contained in the tree is called a trunk. Each branch off the trunk represents the probability that one item in the dataset falls into one of two categories. This branching process continues until the tree can sort the data into categories based on the likelihood of its combined features. The end nodes of a decision tree are known as leaves (Galarnyk, 2022).

For our dataset, the feature set represented includes amount of precipitation, maximum temperature, minimum temperature, wind speed, and type of weather (e.g. rain, snow, sunny), along with the date. During training, the Decision Tree Classifier will start with a random feature and then determine if the next feature is within a certain range. By learning the combinations of features from the dataset, it can predict test data points.

Depth is a measure of how many splits the tree can make before coming to a decision. For Decision Tree Classifier, we tested depths ranging from 1 to 10 as shown in Figure 6 and determined the optimal depth for our model was 3, with a maximum model accuracy of 85.32% using Training dataset and 84.05% for testing dataset, as shown in Figure 7.

```python
# Making predictions and calculating accuracy using Decision tree classifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

depth_range = range(1,11)
param_grid = dict(max_depth = depth_range)
scores = []
for mxdepth in depth_range:
    dt = DecisionTreeClassifier(criterion= 'gini',max_depth =mxdepth,random_state=50)
clf = GridSearchCV(dt,param_grid,cv=5)
grid_search = clf.fit(X_train,Y_train)

print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
clf.fit(X_train,Y_train)
#Make predictions
Y_pred = clf.predict(X_test)
#calculating accuracy
TREE_accuracy = round(accuracy_score(Y_test,Y_pred)*100,2)
# Plotting the confusion matrix on traing and testing data
print(confusion_matrix( Y_test,Y_pred ))
print("The Decision Tree Model accuracy on Test dataset (in %):",TREE_accuracy)
```

Figure 6: Decision Tree classifier code

```
{'max_depth': 3}
Accuracy for our training dataset with tuning is : 85.32%
[[  0   0   0   0  10]
 [  0   0   0   0  41]
 [  0   0 182   1  11]
 [  0   0   7   4   0]
 [  0   0   0   0 183]]
The Decision Tree Model accuracy on Test dataset (in %): 84.05
```

Figure 7: Decision Classifier accuracies for the best depth value

5. RANDOM FOREST

A random forest is a supervised machine learning algorithm that uses iterations of decision trees to determine the most common outcome when all of the trees are examined as a group. The code for the random forest classifier is shown in Figure 8.

```python
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()
param_grid = {'max_features':['sqrt'],
    'max_depth' : [4],
    'criterion' :['gini'],
    'random_state' :[18]}
grid = GridSearchCV(classifier,param_grid, cv =5)
grid_search = grid.fit(X_train,Y_train)
print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
grid.fit(X_train,Y_train)
Y_pred = grid.predict(X_test)
FOREST_accuracy = round(accuracy_score(Y_test,Y_pred)*100,2)
# Plotting the confusion matrix on traing and testing data
print(confusion_matrix( Y_test,Y_pred ))
print("Random forest Classifier model Accuracy on Test dataset (in %): ",FOREST_accuracy)
```

Figure 8: Code for Random Forest classifier

This algorithm can be used for classification as well as regression. Random forests can help prevent overfitting, a common problem with Decision Tree Classifiers, by generating several models and enabling them to make the prediction together. The resulting decision is less dependent on the weaknesses of any individual model, as long as the models are not strongly correlated with each other (Yiu, 2021).

For our random forest model, since weather is already highly random, using the random forest did not significantly alter the accuracy of the model compared to the single Decision Tree. It produced an accuracy of 85.42% using the Training dataset and an accuracy of 83.83% using Testing dataset as shown in Figure 9.

```
{'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'random_state': 18}
Accuracy for our training dataset with tuning is : 85.42%
[[  0   0   0   0  10]
 [  0   0   0   0  41]
 [  0   0 183   0  11]
 [  0   0   9   2   0]
 [  0   0   0   0 183]]
Random forest Classifier model Accuracy on Test dataset (in %):  83.83
```

Figure 8: Code for Random Forest classifi
Figure 9: Accuracies from Random Forest Classifier

6. SUPPORT VECTOR CLASSIFIER

A Support Vector Machine (SVM) is a supervised learning method used for classification, regression, and outlier detection, which is effective in high dimensional spaces. An SVM divides data linearly by determining the best hyperplane between groups of data. It does this by determining the plane with the maximum margin between the groups, based on the points that are closest to the plane. These points are known as support vectors. If the data is not linearly separable, a Kernel function can be used to shift the data into higher dimensionality, enabling a plane to be chosen in that space. A Support Vector Classifier operates similarly to an SVM, but compensates for the shape of the data groups non-linearly (Support Vector Machines, n.d.).

For our weather prediction model, code for which is shown in Figure 10, when we used the Support Vector classifier, it produced an accuracy of 85.22% for Training dataset and an accuracy of 84.05% using the Testing dataset, as shown in Figure 11.

```
from sklearn.svm import SVC
classifier = SVC()
param_grid= {'kernel':['linear','rbf'], 'C':[1,10]}
grid = GridSearchCV(classifier,param_grid, cv =5)
grid_search = grid.fit(X_train,Y_train)
print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
grid.fit(X_train,Y_train)
Y_pred = grid.predict(X_test)
SVC_accuracy = round(accuracy_score(Y_test,Y_pred)*100,2)
# Plotting the confusion matrix on traing and testing data
print(confusion_matrix( Y_test,Y_pred ))
print("Support Vector Classifier model Accuracy on Test dataset (in %): ",SVC_accuracy)
```

Figure 10: Code for Support Vector Classifier

```
{'C': 1, 'kernel': 'linear'}
Accuracy for our training dataset with tuning is : 85.22%
[[  0   0   0   0  10]
 [  0   0   0   0  41]
 [  0   0 183   0  11]
 [  0   0   8   3   0]
 [  0   0   0   0 183]]
Support Vector Classifier model Accuracy on Test dataset (in %):  84.05
```

Figure 11: Accuracies from Support Vector Classifier

7. STOCHASTIC GRADIENT DESCENT (SGD)

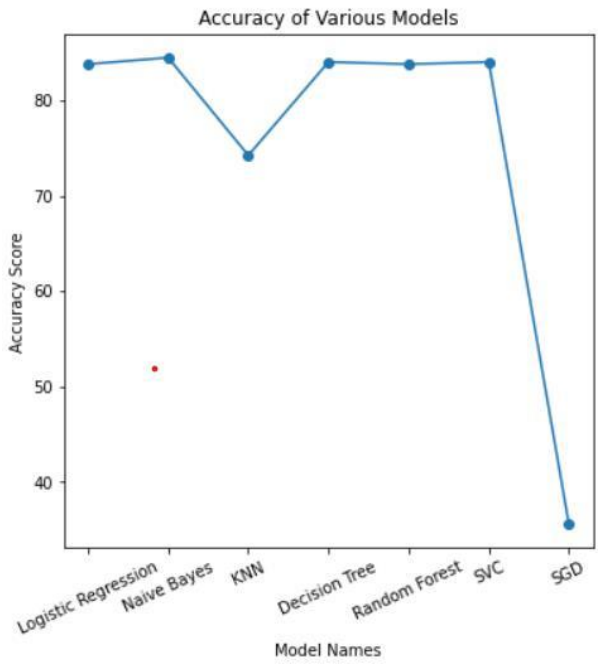
Stochastic Gradient Descent is

```
from sklearn.linear_model import SGDClassifier
classifier = SGDClassifier()
param_grid= {'loss':['squared_error'], 'penalty':['l2'], 'max_iter':[300],'tol':[10]}
grid = GridSearchCV(classifier,param_grid, cv =5)
grid_search = grid.fit(X_train,Y_train)
print(grid_search.best_params_)
accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
grid.fit(X_train,Y_train)
Y_pred = grid.predict(X_test)
SGD_accuracy = round(accuracy_score(Y_test,Y_pred)*100,2)
# Plotting the confusion matrix on traing and testing data
print(confusion_matrix( Y_test,Y_pred ))
print("Stochastic Gradient Descent Classifier model Accuracy on Test dataset (in %): ",SGD_accuracy)
```

```
{'loss': 'squared_error', 'max_iter': 300, 'penalty': 'l2', 'tol': 10}
Accuracy for our training dataset with tuning is : 17.34%
[[  0   0  10   0   0]
 [  0   0  41   0   0]
 [  0  38 156   0   0]
 [  2   6   3   0   0]
 [  1   0 180   2   0]]
Stochastic Gradient Descent Classifier model Accuracy on Test dataset (in %):  35.54
```

7. COMPARISON OF ALGORITHMS

The Algorithms Logistic Regression, Naïve Bayes classifier, KNN Classifier, Decision Tree Classifier, Random Forest Classifier, Support Vector Classifier were implemented for prediction of Seattle Weather using Training and Testing datasets with split of 70% Training and 30% Testing dataset. The Algorithm that performed best with highest accuracy among these classifier



algorithms is Naïve Bayes classifier with an accuracy of 84.51% as shown in Figure 12.

Figure 12: Accuracies comparison for all classifiers

7. CONVOLUTIONAL NEURAL NETWORK

A major component and class of artificial intelligence is a convolutional neural network (CNN). A CNN is a deep learning algorithm that takes an image as an input and assign weights and biases to it to differentiate it from other images. It extracts a set of features based on the input image while maintaining relationships with the neighboring pixels. CNN contains multiple layers such as convolution layers, pooling layers, and dense layers. Additionally, it contains activation layers and fully connected layers. The different operations involved in a CNN include Convolution, pooling, flatten and classification operation (Saha, 2021).

CNN has been used as a major example in this paper and can work well for financial automation tasks.

8. FEEDFORWARD NEURAL NETWORK

The feed forward neural network was the initial and foundational neural network created. Therefore, it is the simplest type of neural network. Although this can have many

advantages, including problems that require step functions, the other types of neural networks, such as the convolutional neural network, are better suited for a weather problem such as the one focused on in this paper.

9. CONCLUSION

Of the algorithms and training parameters tested, the one that predicted Seattle's weather with the greatest accuracy was the Naïve Bayes Classifier model, with an accuracy of 84.51%. Modifications to the model that would be worth trying to improve the accuracy in the future include treating outlier values and experimenting with different algorithms than the one included in our Sci-Kit Learn library. Attempting to train on additional data may or may not be helpful, given the impacts of climate change on regional weather patterns in the recent past; however, if additional, comparably obtained data can be added to the model, it is worth testing.

10. REFERENCES

Support Vector Machines. (n.d.). Scikit-Learn. Retrieved August 21, 2022, from https://scikit-learn.org/stable/modules/svm.html

Brownlee, J. (2020, August 15). Naive Bayes for Machine Learning. Machine Learning Mastery. Retrieved August 21, 2022, from https://machinelearningmastery.com/naive-bayes-for-machine-learning/

Galarnyk, M. (2022, April 27). *Understanding Decision Trees for Classification (Python)*. Medium. Retrieved August 21, 2022, from https://towardsdatascience.com/understanding-decision-trees-for-classification-python-9663d683c952

Hrouda-Rasmussen, S. (2022, March 30). How (Gaussian) Naive Bayes Works | Towards Data Science. Medium. Retrieved August 21, 2022, from https://towardsdatascience.com/gaussian-naive-bayes-4d2895d139a

Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms (1st ed.). Manning Publications. https://learning.oreilly.com/videos/grokking-artificial-intelligence/9781617296185AU/

IBM Integrated Analytics System. (2021, December 17). Background of KNN. IBM. Retrieved August 21, 2022, from https://www.ibm.com/docs/en/ias?topic=knn-background

Kaggle. (2022, January 17). WEATHER PREDICTION (1.0) [Seattle-weather.csv]. Kaggle. https://www.kaggle.com/ananthr1/weather-prediction

Saha, S. (2021, December 7). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Rothman, D. (2018). Artificial Intelligence By Example: Develop machine intelligence from scratch using real artificial intelligence use cases (2nd ed.). Van Haren Publishing.

Yiu, T. (2021, December 10). Understanding Random Forest - Towards Data Science. Medium. Retrieved August 21, 2022, from https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Thanda, A. (2022, May 24). What is Logistic Regression? A Beginner's Guide. Career Foundry. https://careerfoundry.com/en/blog/data-analytics/what-is-logistic-regression/#what-is-logistic-regression