Introduction to Data Mining

DATS 6103 – Spring 2020

Group 7 Final Report

US Reported Accidents (Visualization and Summary)

Submitted by:

Daniel Frey

Gregg Legarda

Siwei Wang

Submitted to:

Dr. Amir Jafari (PhD.)

INTRODUCTION

This project investigates the "US_Accidents_Dec19.csv" file from Kaggle, a dataset with nearly 3 million observations of car accidents in the United States from February 2016 to December 2019. As a means of analysis, this project features a graphic user interface (GUI) that employs data mining techniques such as preprocessing, cleaning, data analysis and modeling. Models include decision tree, random forest, KNN, support vector machine, Naive Bayes, and logistic regression. Using accident "severity" as our target variable, can our models accurately predict severity based on feature variables such as weather conditions?

This report will be split among several sections, including: 1) a description of the dataset; 2) a theoretical overview of the algorithms used; 3) the processes we used to prepare the data for modeling; 4) the results of the models themselves; and 5) a summary and conclusion of our report.

DESCRIPTION OF THE DATASET

The dataset we used features nearly 3 million records of car accidents in the U.S. from February 2016 to December 2019. It houses 49 features, ranging from categorical to numerical data types. The "severity" variable was chosen as our target for the machine learning models, since it is categorical and, in theory at least, would seem to be a dependent variable. Severity "shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay)." Feature variables, on the other hand, included: distance ("The length of the road extent affected by the accident"), temperature "Shows the temperature (in Fahrenheit)", wind chill ("Shows the wind chill (in Fahrenheit)"), humidity ("Shows the humidity (in percentage)"), pressure ("Shows the air pressure (in inches)"), visibility ("Shows visibility (in miles)"), wind speed "Shows wind speed (in miles per hour)", and precipitation "Shows precipitation amount in inches, if there is any".

ALGORITHMS

KNN

The k-nearest neighbors algorithm, or KNN, classifies data points based on those data points' neighbors' classification. On the whole, KNN is often chosen for its simplicity and ease of use. Nearest neighbors are determined by distance metrics; Euclidean, Minkowski, and Manhattan distances are often used in determining data points' nearest neighbors.

Manhattan Distance[1]:

$$d(i,j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}$$

Euclidean Distance[2]:

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}$$

[1] Equations from https://www.sciencedirect.com/topics/computer-science/minkowski-distance
[2] Ibid.

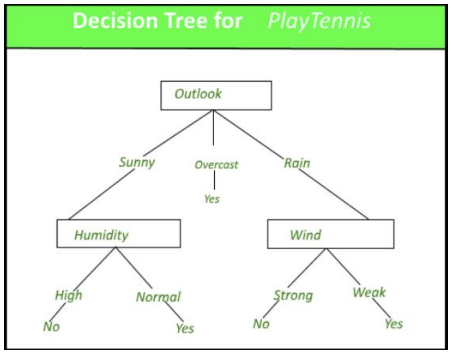
Minkowski Distance[3]:

$$d_1(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

To cite an example, if “k” is set to 7 neighbors, and a data point is surrounded by 4 red points and 3 greens, the data point would be classified as red, since reds outweigh the greens. (K is typically set to an odd number to avoid ties.) KNN’s accuracy is affected by the value one selects for “k”; to determine the best value for “k”, one must run the algorithm over multiple values of “k” and determine which offers the lowest error rate. Error rate, however, is not the only consideration when determining “k”; there is also the tradeoff of using “small” or “large” k-values; generally speaking, smaller values cause the KNN algorithm to be more sensitive to noise, and large k-values become computationally expensive.[4]

Decision Tree

In short, “A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or [continuous] value).”[5] See the following figure for a visualization of a typical decision tree[6]:



The rules themselves are based on metrics called “Gini index” and “entropy.” Which metric is used depends on the specific decision tree algorithm implemented (Iterative Dichotomiser, or ID3, uses entropy/information gain; CART, or Classification and Regression Tree, uses Gini); Gini is generally less computationally expensive. Gini values measure “the degree or probability of a particular variable being wrongly classified when it is randomly chosen.”[7] Entropy, meanwhile, measures the amount of information gained from features; lower entropy yields higher information gain.

[3] Ibid.
[4] https://askdatascience.com/170/what-happens-asthe-k-increases-in-the-knn-algorithm
[5] https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1
[6] https://www.geeksforgeeks.org/decision-tree/
[7] https://blog.quantinsti.com/gini-index/#Gini-Index

Gini[8]

$$1 - \sum_{i=1}^{n} (p_i)^2$$

Entropy[9]

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

The feature with the lowest Gini or entropy value becomes the root (top) node; this node houses the most important feature, as all other decisions stem from it. In this way, decision trees perform a kind of automatic feature selection. The final leaves, meanwhile, represent the target variable's classification; for the data's test set, these would represent the predictions.

Random Forest[10]

The random forest algorithm follows the same principles as the decision tree model, but it also uses ensembling. Ensembling employs multiple models at once to produce more accurate results. Random forest essentially runs multiple decision tree models at once, and takes a majority vote of the results for better accuracy. The user ultimately determines the number of trees to create.

Logistic Regression (Multinomial/Multiclass)[11]

Multiclass logistic regression can be applied to a dataset when the target variable is categorical and holds more than 2 classes. For each class within the target variable (in our case, "severity"), the algorithm effectively creates a binary logistic regression model, which is based on probability. On any given new data point (i.e. the new data we want our algorithm to classify), the logistic regression model will pick the class model that maximizes the probability of having correctly identified the class; the class with the highest calculated probability becomes the predicted class for the new data point.

Naive Bayes[12]

Naive Bayes is a generative algorithm, meaning that it learns by "inquiring" about the features of the target variable's classes. The mathematical foundation of the Naive Bayes classifier is the Bayes theorem, copied below.

Bayes Theorem[13]

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

---

[8] Ibid.
[9] Ibid.
[10] https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725
[11] https://it.unt.edu/sites/default/files/mlr_jds_aug2011.pdf; https://www.youtube.com/watch?v=DuXmFxDSc9U
[12] https://www.youtube.com/watch?v=z5UQyCESW64
[13] https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

According to the theorem, once can determine the probability of A (the hypothesis) given B (the evidence), assuming that the features are all independent of each other.[14] In reality, features rarely exhibit zero correlation with each other; as a result, multicollinearity may affect the performance of the Naive Bayes classifier.[15]

Support Vector Machine (SVM)[16]

SVM models attempt to find a decision boundary (known formally as a hyperplane) that maximizes the distance between the classes of data; the hinge loss function performs this maximization (see equation below).

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases} \qquad c(x, y, f(x)) = (1 - y * f(x))_+$$

The data points closest to the hyperplane are the ones that most strongly "influence the position and orientation of the hyperplane."[17] The number of features in the model affects the dimensionality of the hyperplane; an SVM model with two features has a 2-dimensional hyperplane (a line), while a model with three features has a 3-dimensional hyperplane (a plane, such as a rectangle). Higher dimensional models are harder to visualize.

EXPERIMENTAL SETUP

For preprocessing, we first removed unnecessary columns from the dataset. These included non-numeric data, such as categorical types; traffic pattern and location data constituted the majority of this data. We defined the variables of interest with the following code from the "train_model.py" script:

```
#======= SPLIT FEATURES AND TARGET ==========#
column_names = ["Distance(mi)", "Temperature(F)", "Wind_Chill(F)", "Humidity(%)", "Pressure(in)",
                "Visibility(mi)", "Wind_Speed(mph)", "Precipitation(in)",
                "Severity"]
X = [list(self.data["Distance(mi)"]), list(self.data["Temperature(F)"]),
     list(self.data["Wind_Chill(F)"]), list(self.data["Humidity(%)"]), list(self.data["Pressure(in)"]),
     list(self.data["Visibility(mi)"]), list(self.data["Wind_Speed(mph)"]), list(self.data["Precipitation(in)"])]
X = np.transpose(X)
y = list(self.data["Severity"])
```

[14] Ibid.
[15] Ibid.
[16] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
[17] Ibid.

We removed unnecessary columns from the dataset with this code from the “pre_process.py” script:

```
def cleanup_data(self):
    #########------------------------------------ DATA CLEANUP ------------------------------------ #########
    print("Cleaning Dataframe...")
    #delete uneccessary columns
    del self.data['ID']
    del self.data['Source']
    del self.data['TMC']
    del self.data['End_Lat']
    del self.data['End_Lng']
    del self.data['Country']
    del self.data['Timezone']
    del self.data['Airport_Code']
    del self.data['Weather_Timestamp']
    del self.data['Wind_Direction']

    del self.data['Weather_Condition']
```

We then cleaned the dataset by changing data types and filling in missing data (also in the “pre_process.py” script):

```
#########------------------------------------ FILL MISSING VALUES ------------------------------------ #####

# fill other predict attributes with means of columns
self.data['Start_Lat'] = self.data['Start_Lat'].fillna(self.data['Start_Lat'].mean())
self.data['Start_Lng'] = self.data['Start_Lng'].fillna(self.data['Start_Lng'].mean())
self.data['Distance(mi)'] = self.data['Distance(mi)'].fillna(self.data['Distance(mi)'].mean())
self.data['Number'] = self.data['Number'].fillna(self.data['Number'].mean())
self.data['Temperature(F)'] = self.data['Temperature(F)'].fillna(self.data['Temperature(F)'].mean())
self.data['Wind_Chill(F)'] = self.data['Wind_Chill(F)'].fillna(self.data['Wind_Chill(F)'].mean())
self.data['Humidity(%)'] = self.data['Humidity(%)'].fillna(self.data['Humidity(%)'].mean())
self.data['Pressure(in)'] = self.data['Pressure(in)'].fillna(self.data['Pressure(in)'].mean())
self.data['Visibility(mi)'] = self.data['Visibility(mi)'].fillna(self.data['Visibility(mi)'].mean())
self.data['Wind_Speed(mph)'] = self.data['Wind_Speed(mph)'].fillna(self.data['Wind_Speed(mph)'].mean())
self.data['Precipitation(in)'] = self.data['Precipitation(in)'].fillna(self.data['Precipitation(in)'].mean())
self.data['Severity'] = self.data['Severity'].fillna(self.data['Severity'].mode())

# fill address with filler
self.data['Description'] = self.data['Description'].fillna(0)
self.data['Street'] = self.data['Street'].fillna(0)
self.data['City'] = self.data['City'].fillna(0)
self.data['State'] = self.data['State'].fillna(0)
self.data['Zipcode'] = self.data['Zipcode'].fillna(0)
```

```
#########------------------------------------ CHANGE DATA TYPES ------------------------------------ #########
self.data['Description'] = np.where(pd.isnull(self.data['Description']),self.data['Description'],self.data['Description'].astype(str))
self.data['Street'] = np.where(pd.isnull(self.data['Street']),self.data['Street'],self.data['Street'].astype(str))
self.data['City'] = np.where(pd.isnull(self.data['City']),self.data['City'],self.data['City'].astype(str))
self.data['State'] = np.where(pd.isnull(self.data['State']),self.data['State'],self.data['State'].astype(str))
self.data['Zipcode'] = np.where(pd.isnull(self.data['Zipcode']),self.data['Zipcode'],self.data['Zipcode'].astype(str))
```

Moreover, we standardized our data and performed principal component analysis (PCA) before running some of our models to account for different orders of magnitude and high-dimensionality, respectively.

```
# ======= PERFORM STANDARD SCALER ==========#
#for DT, RF, LR,
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
```

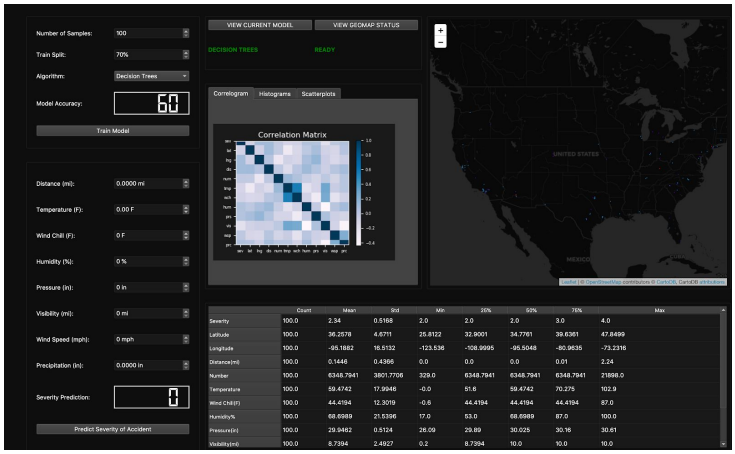
```
#======= PERFORM PCA ==========#
#for KNN and Logistic Regression
if self.train_inputs[2] == "KNN" or self.train_inputs[2] == "Logistic Regression":
    X_train = self.pca.fit_transform(X_train)
    X_test = self.pca.fit_transform(X_test)
    print("PCA PERFORMED")
    print("PCA model explained variance\n", self.pca.explained_variance_)
    print("PCA model explained variance ratio \n", self.pca.explained_variance_ratio_)
    print("PCA model explained variance ratio cumsum\n", self.pca.explained_variance_ratio_.cumsum())
```

We also implemented Grid Search CV with many of the models (KNN, decision tree, random forest, and support vector machine) to optimize their hyperparameters and ultimately, to improve

model accuracy. See the image below for an example of grid search implementation with the decision tree algorithm.

```
###### GRID SEARCH  #######
clf_gs = tree.DecisionTreeClassifier()
parameter_grid = {'criterion': ['gini', 'entropy'],  # provides better parameters for model above
                  'splitter': ['best', 'random'],
                  'max_depth': [1, 2, 3],
                  'max_features': [1, 2, 3, 4]}
grid_search = GridSearchCV(clf_gs, param_grid=parameter_grid, cv=10)
grid_result = grid_search.fit(X_train, y_train)
best_params = grid_result.best_params_
print(best_params)
best_clf = tree.DecisionTreeClassifier(criterion=best_params['criterion'],splitter=best_params['splitter'],
                                       max_depth=best_params['max_depth'], max_features=best_params['max_features'],random_state=23)
best_clf.fit(X_train,y_train)
# Predict test from grid search
y_pred_grid = best_clf.predict(X_test)
print("GS shape of y_pred:", X_test.shape)
acc_score_gs = accuracy_score(y_test, y_pred_grid)
self.accuracy = (acc_score_gs * 100).round(2)
# Accuracy
print('GS DT Accuracy:', acc_score_gs)
print('GS DT Accuracy:', self.accuracy)
self.model_algorithm = best_clf
print("GS train model is:", best_clf)
```

As for the models themselves, implementing them within the GUI itself is relatively straightforward. The user selects the desired train/test split and the number of random samples to take, clicks on a drop-down menu, selects the desired model, and clicks the “Train Model” button. The app then runs the sample through the chosen algorithm. Once training is complete, an accuracy reading is displayed, showcasing the percentage of correct classifications the algorithm made in the test set. The user can then input values for the feature variables and predict a severity level based on that input.

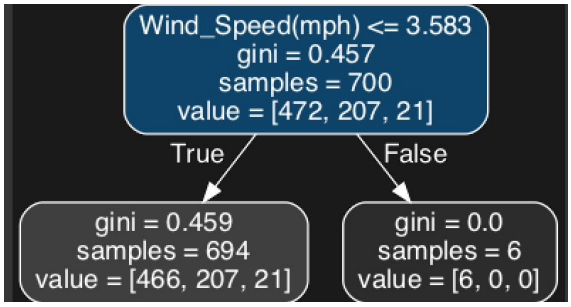


RESULTS

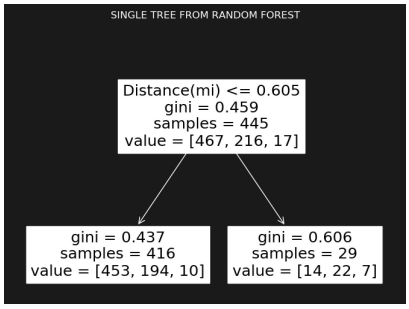
N=1000, Train Split: 70%

Model	Accuracy
Decision Tree	64.67%
Random Forest	66%
Logistic Regression	64%
K-Nearest Neighbors (KNN)	63.33%
Support Vector Machine (SVM)	71%
Naive Bayes	39.33%

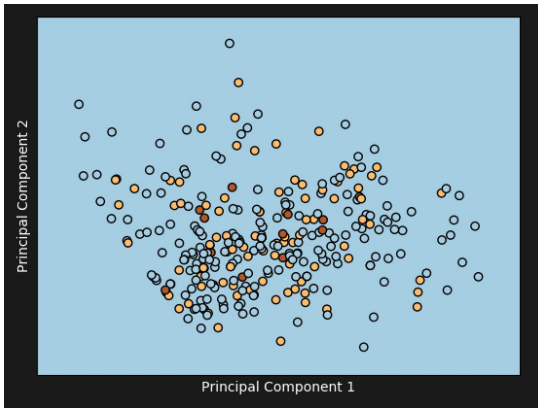
Decision Tree:



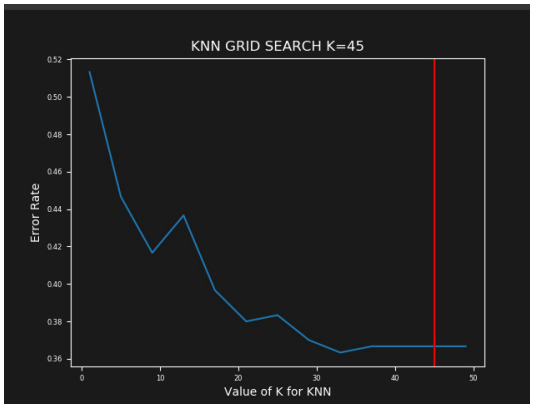
Random Forest:



Logistic Regression:



KNN:

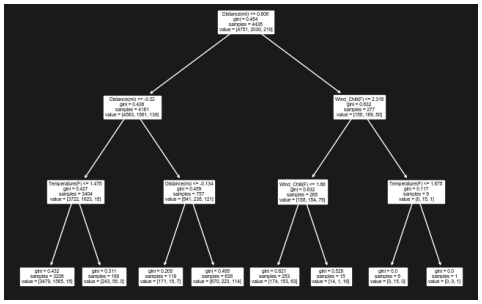


In terms of feature selection, our decision tree model chose wind speed as the most important variable of interest, while the random forest chose distance. At the 1000 sample level, support vector machine boasts the highest accuracy, at 71%.

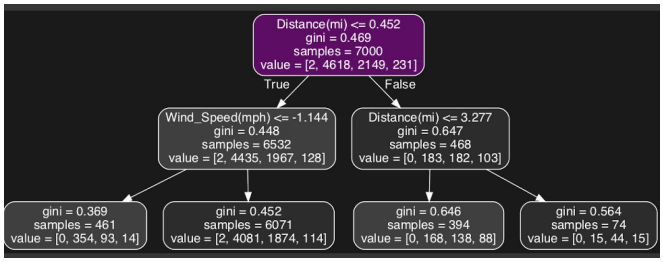
N=10,000, Train Split: 70%

Model	Accuracy
Decision Tree	66%
Random Forest	66.47%
Logistic Regression	66.53%
K-Nearest Neighbors (KNN)	67%
Support Vector Machine (SVM)	68.8%
Naive Bayes	55.9%

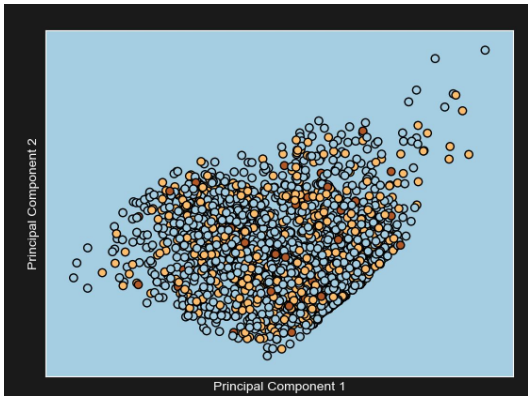
Random Forest



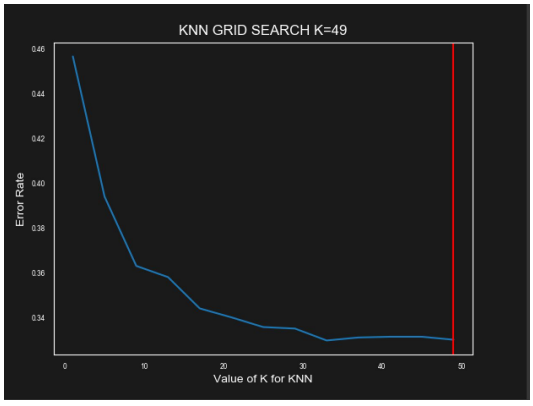
Decision Tree



Logistic Regression



KNN



At the 10,000 sample level, our random forest model (at least the tree we chose to highlight) and decision tree model chose the “distance” variable as the most important. Support vector machine again proved the most accurate, at 68.8% correct classification.

SUMMARY & CONCLUSIONS

This project showcases the machine learning, model-building process. With the appropriate formatting, cleaning, and pre-processing, data can be harnessed for machine learning models. With appropriate parameter tuning and result validation, machine learning models can produce accurate predictions based on the training data. Determining which model is most appropriate is highly case-dependent, and depends largely on the attributes of the data itself. Is the data highly dimensional? Is it heavily skewed? Does it house numerical or categorical features? These are some of the questions one must consider when deciding on a model.

In our case, at both the 1,000 and 10,000 sample levels, support vector machine proved the most accurate (71% and 68.8%, respectively). Applying preprocessing techniques to the dataset, including gridsearch, PCA, and standardization, improved accuracy; increasing the sample size also proved beneficial to accuracy levels.

For the future, we might consider ensembling our models together; including categorical features as part of the dataset to be evaluated by our models; evaluating our features for multicollinearity; and applying the log transform method to the “severity” target variable to account for its skew.

REFERENCES

Chen, L. (n.d.). Basic Ensemble Learning (Random Forest, AdaBoost, Gradient Boosting)- Step by Step Explained. *Towards Data Science*. https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725

Decision Tree. (n.d.). https://www.geeksforgeeks.org/decision-tree/

Gandhi, R. (2018a, May 5). *Naive Bayes Classifier*. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Gandhi, R. (2018b, June 7). Support Vector Machine – Introduction to Machine Learning Algorithms. *Towards Data Science*. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Ng, A. (2015, February 10). *Andrew Ng Naive Bayes Generative Learning Algorithms*. https://www.youtube.com/watch?v=z5UQyCESW64

Ng, A. (2017, August 9). *Logistic Regression—Multiclass Classification One vs all*. https://www.youtube.com/watch?v=DuXmFxDSc9U

Pei, J., & Han, J. (2012). Minkowski Distance. In *Data Mining* (3rd ed.). https://www.sciencedirect.com/topics/computer-science/minkowski-distance

Sanjeevi, M. (2017, October 6). *Chapter 4: Decision Trees Algorithms*. https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1

Starkweather, J., & Moske, A. (n.d.). *Multinomial Logistic Regression*. University of North Texas. https://it.unt.edu/sites/default/files/mlr_jds_aug2011.pdf

Tahsildar, S. (2019, April 18). *Gini Index For Decision Trees*.

https://blog.quantinsti.com/gini-index/#Gini-Index

What happens as the K increases in the KNN algorithm? (n.d.). Ask Data Science!

https://askdatascience.com/170/what-happens-asthe-k-increases-in-the-knn-algorithm