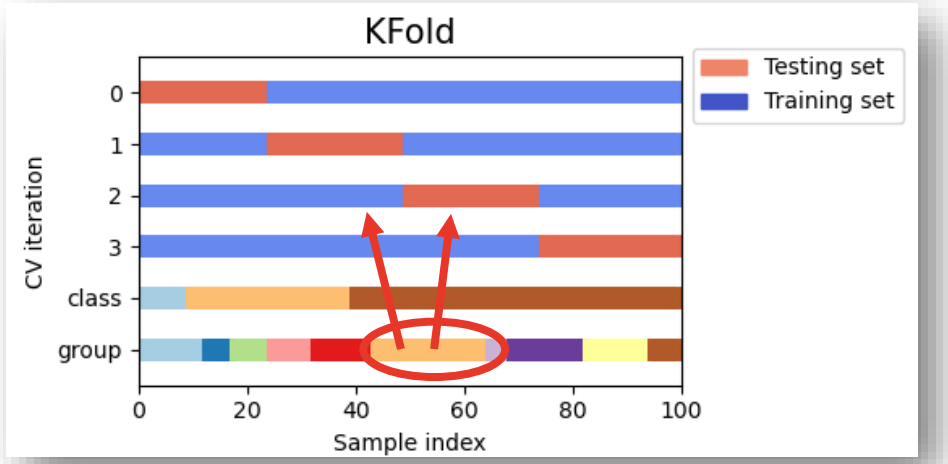
CLASSIFICATION

Team: The Major
Carson Wittwer

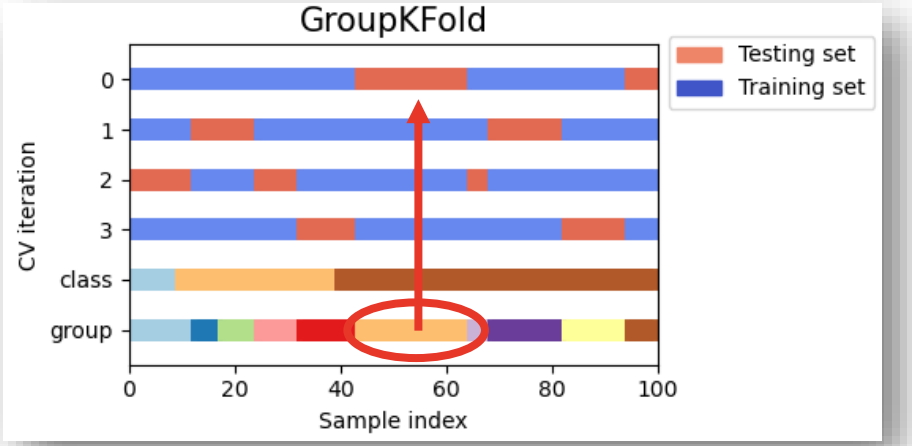
HOW WAS THE CROSS-VALIDATION SPLIT DONE, AND WHY?

- Our question is based on predicting the quadrant of a clip, so we want to ensure that we aren't training our models on the same clip that we will test
- Quadrants are classified to a unique *pianist_id* and *segment_id* combination(*PSid*)
- The data is setup to further break down these *PSid*s into *snippets*, which all have the same quadrant class, but allow additional information to be fed into our models
  - Each *PSid* is the same clip of music, so can share many similarities across *snippets*
- To ensure there is no data leakage, we need to contain all the *snippets* of a *Psid* into either the train or test groups, therefore we must group by *PSid* when selecting our train and test data
- To accomplish this, using SKLearn utilities, we want to use variations of a folding validation where grouping is used
  - I.E. GroupKFold, StratifiedKFold, GroupShuffleSplit, etc



Portion of Group in both Test and Training Set

Entirety of Group in both Test and Training Set



Data Leakage:

In statistics and machine learning, leakage (also known as data leakage or target leakage) is the use of information in the model training process which would not be expected to be available at prediction time, causing the predictive scores (metrics) to overestimate the model's utility when run in a production environment. [1]

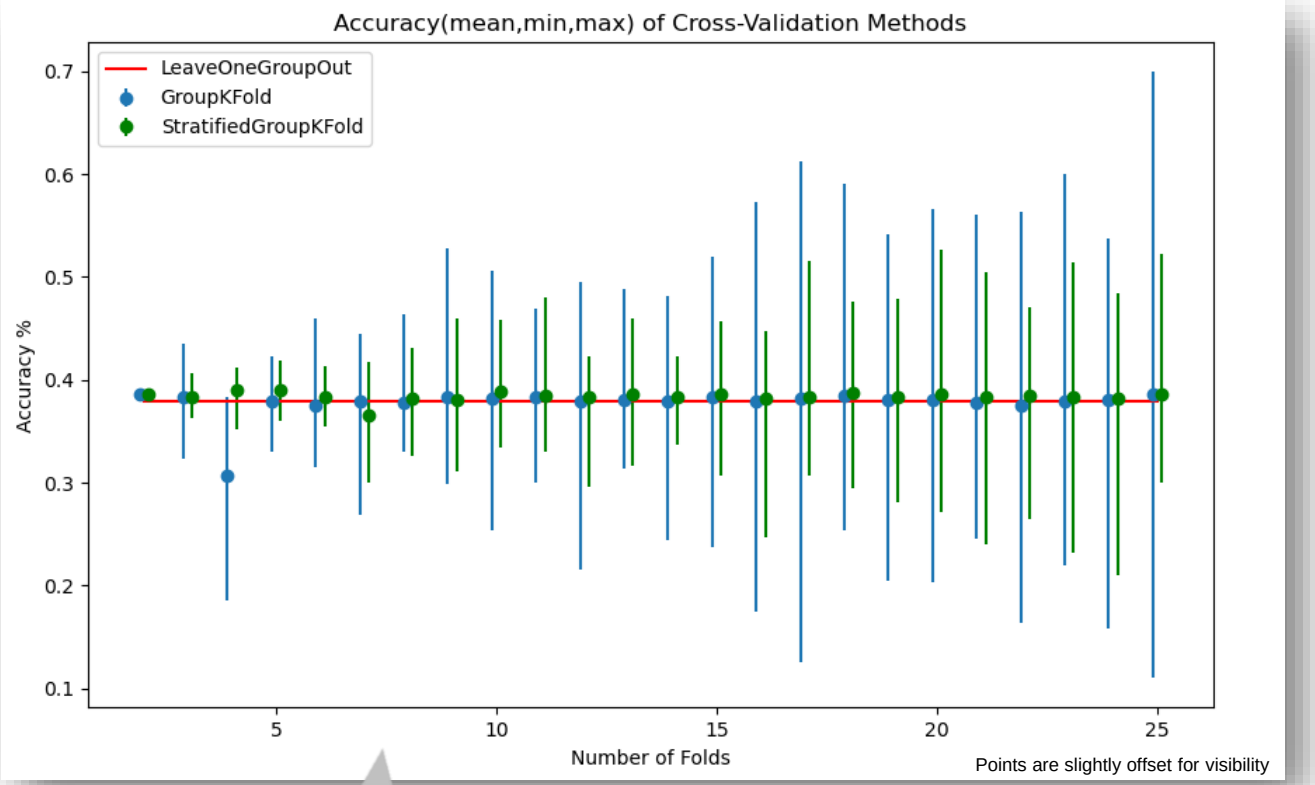
[1] Shachar Kaufman, Saharon Rosset, and Claudia Perlich. 2011. Leakage in data mining: formulation, detection, and avoidance. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '11). Association for Computing Machinery, New York, NY, USA, 556–563. https://doi.org/10.1145/2020408.2020496

Graphics from scikit-learn

JKU

HOW WAS THE CROSS-VALIDATION SPLIT DONE, AND WHY?

- There are multiple ways in the SKLearn toolkit to perform the cross-validation with grouping as discussed in the previous slides. Below will look at the performance of two used with this project: GroupKFold and StratifiedGroupKFold
- Per SKLearn, the difference in these two methods is: [1]
  - GroupKFold attempts to create balanced folds such that the number of distinct groups is approximately the same in each fold
  - StratifiedGroupKFold attempts to create folds which preserve the percentage of samples for each class as much as possible given the constraint of non-overlapping groups between splits
- Additionally, LeaveOneGroupOut, another method of splitting data is used as another comparison
  - This functions basically the same as GroupKFold where the number of folds is equal to 1
- A default SKLearn NaiveBayes model was used to obtain performance measures
  - This can give a general direction of which method is performing better, but may not be true for all models



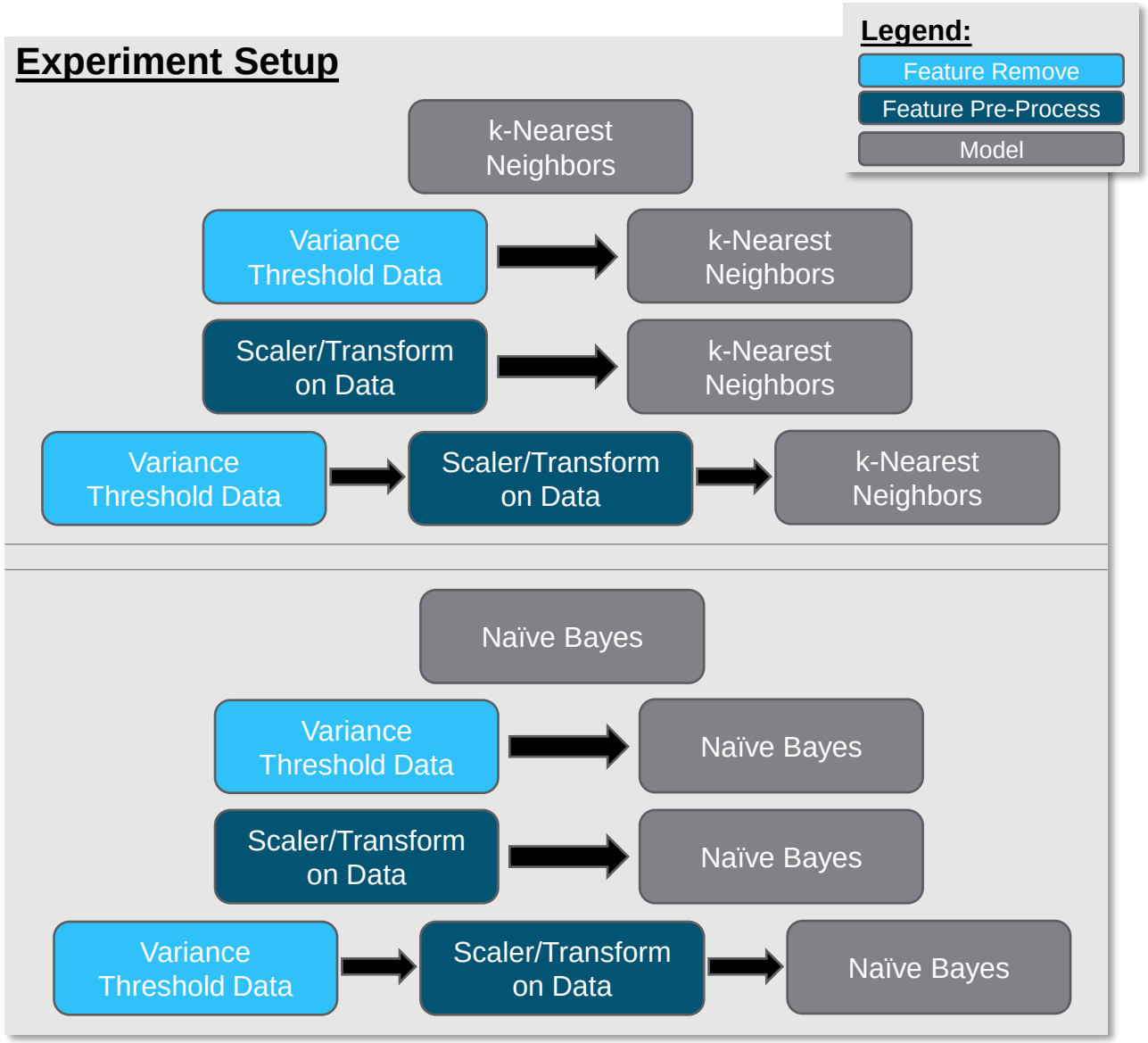
Analysis Outcomes:

- The chart above seems to show that the general performance is best in the range of **4-8 folds**
- **StratifiedGroupKFold** method seems to perform more consistently to the mean
- GroupKFold produced both the most and least accurate single folds(higher maxs and lower mins)

[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedGroupKFold.html

WHICH SUBSET OF FEATURES WAS SELECTED, OR WHICH PREPROCESSING WAS APPLIED, AND WHY?

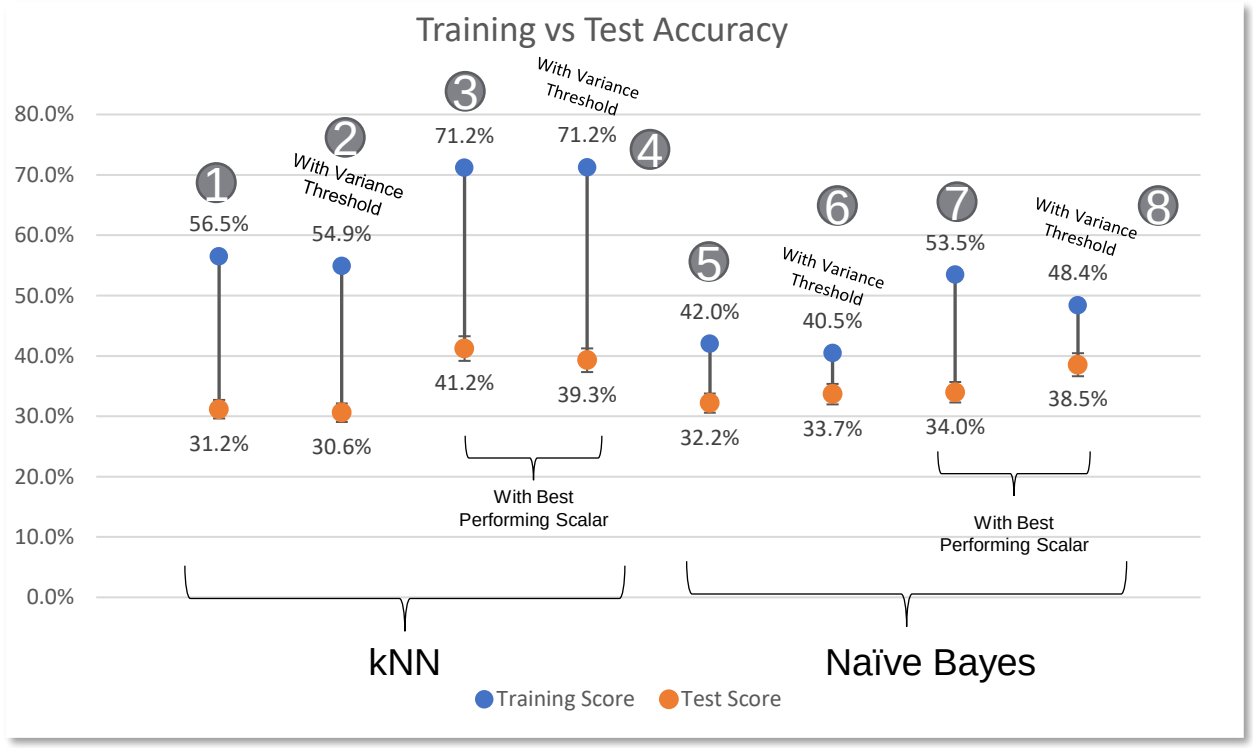
- Per parameter constraints of the assignment, initial experimentation **only** includes **Low** and **Mid** Level features
- Another potential avenue of removing features is using a Variance Threshold, available in the SKLearn toolkit
  - Variance Thresholds removes features with no(constants) or low variance
  - We tested variance of 0.05, which removed 77 of the 169 total features with the lowest variance
- [1] For testing different preprocessing steps, we tested 4 different pre-processing steps in SKLearn that all work to normalize or transform the features
  - StandardScalar, MinMaxScaler, Normalizer, MaxAbsScaler
- Experiment setup including using a default Naïve Bayes and kNN, with and without Variance Thresholds, and identifying the best Scaler/Transformer preprocessing
  - To determine the best Scaler/Transformer, the models where both tested with GridSearchCV available in SKLearn



1. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

WHICH SUBSET OF FEATURES WAS SELECTED, OR WHICH PREPROCESSING WAS APPLIED, AND WHY?

- The chart too the right depicts the outcomes of training and test accuracy for our setups described on the previous slide
- First immediate notice is the increase in performance we see from implementing a Scalar/Transform to the data
  - Comparisons between 1 & 2 versus 3 & 4 show significant performance improvement
  - Comparisons between 4 & 5 versus 6 & 7 show slight but noticeable improvement
- Feature removal through a Variance Threshold slightly hinders performance with kNN and slightly improves performance with Naïve Bayes
  - The best performing scalar was different for the two kNN variants
    - MaxAbsScaler without Variance Thresholding and StandardScalar with Variance Thresholding
  - Both runs, the best performing scalar for Naïve Bayes was the StandardScalar
- For model testing, based on this experiment, we will move forward with using the StandardScalar. VarianceThreshold may be used, but will be model dependent



Not directly pertinent to feature selection, but an immediate visual conclusion we can see is the significant performance gap between kNN training score and the decrease to its test score. This denotes some significant overfitting and generalization issues compared to the Naïve Bayes performance

Standard Scalar:

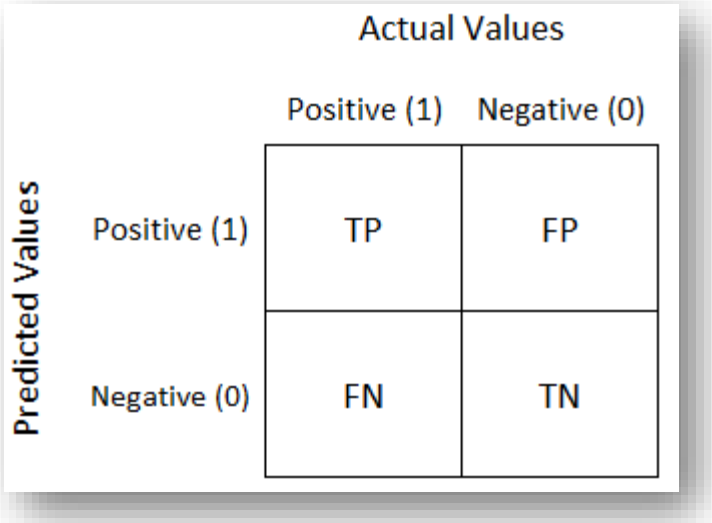
Standardize features by removing the mean and scaling to unit variance.[1]

1. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler

WHAT WOULD BE AN APPROPRIATE EVALUATION CRITERION TO COMPARE PARAMETER SETTINGS AND ALGORITHMS?

- Our main goal is to accomplish classification, provided a series of different data points, so our evaluation metrics should focus on identifying performance of classification

Fundamentally, all the metrics we are interested in come from the **Confusion Matrix:** [1]



TP: True Positive
FP: False Positive
FN: False Negative
TN: True Negative

Accuracy:

- The ratio of correct predictions to total number of predictions
- Best for well balanced classes

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$ [2]

Precision:

- The ratio of True Positives to all positives predicted
- Better for unbalanced classes

$$Precision = \frac{TP}{TP + FP}$$ [2]

Recall:

- The ratio of true positives to all positives (true positives and false negatives) in the dataset.

$$TPR = Recall = \frac{TP}{TP + FN}$$ [2]

F1-Score:

- A combination of both Precision and Recall, the higher the number, the better the performance.
- Solid overall metric, especially with unbalanced classes

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$ [2]

1. https://medium.com/@aatish_kayyath/confusion-matrix-lets-clear-this-confusion-4b0bc5a5983c
2. https://medium.com/analytics-vidhya/evaluation-metrics-for-classification-models-e2f0d8009d69

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

MODEL TESTING SETUP

Given previous learnings discussed in prior slides, our setup will include:

- All features from the defined Low and Mid-level feature sets
  - No VarianceThreshold initially, but could be used at a later point
- Preprocessing from the StandardScalar[1] function in SKLearn
  - A method to standardize features by removing the mean and scaling to unit variance
- Evaluation will be performed by looking at 4 metrics
  - Accuracy, Precision, Recall, and F1 Score

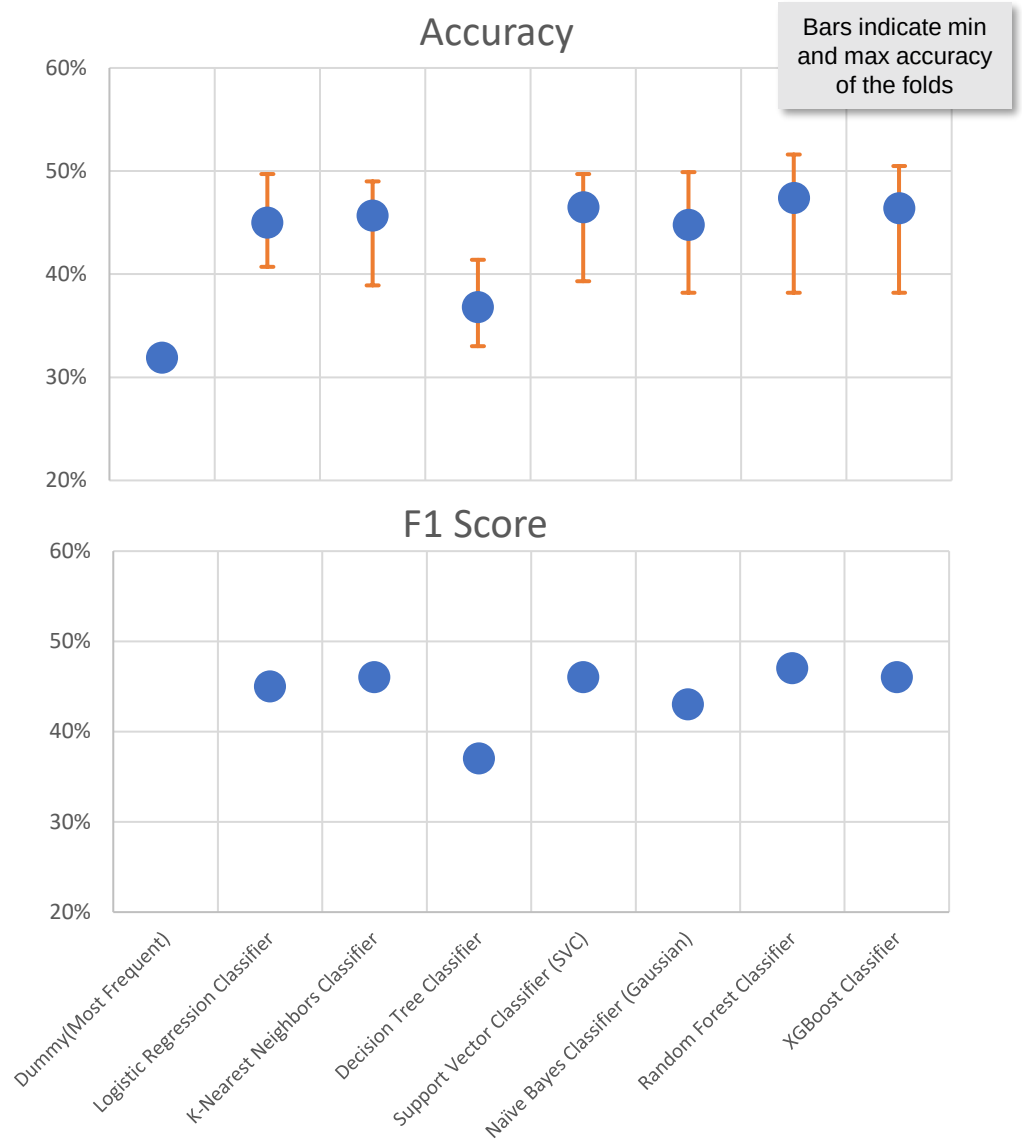
Learning Algorithms to Test

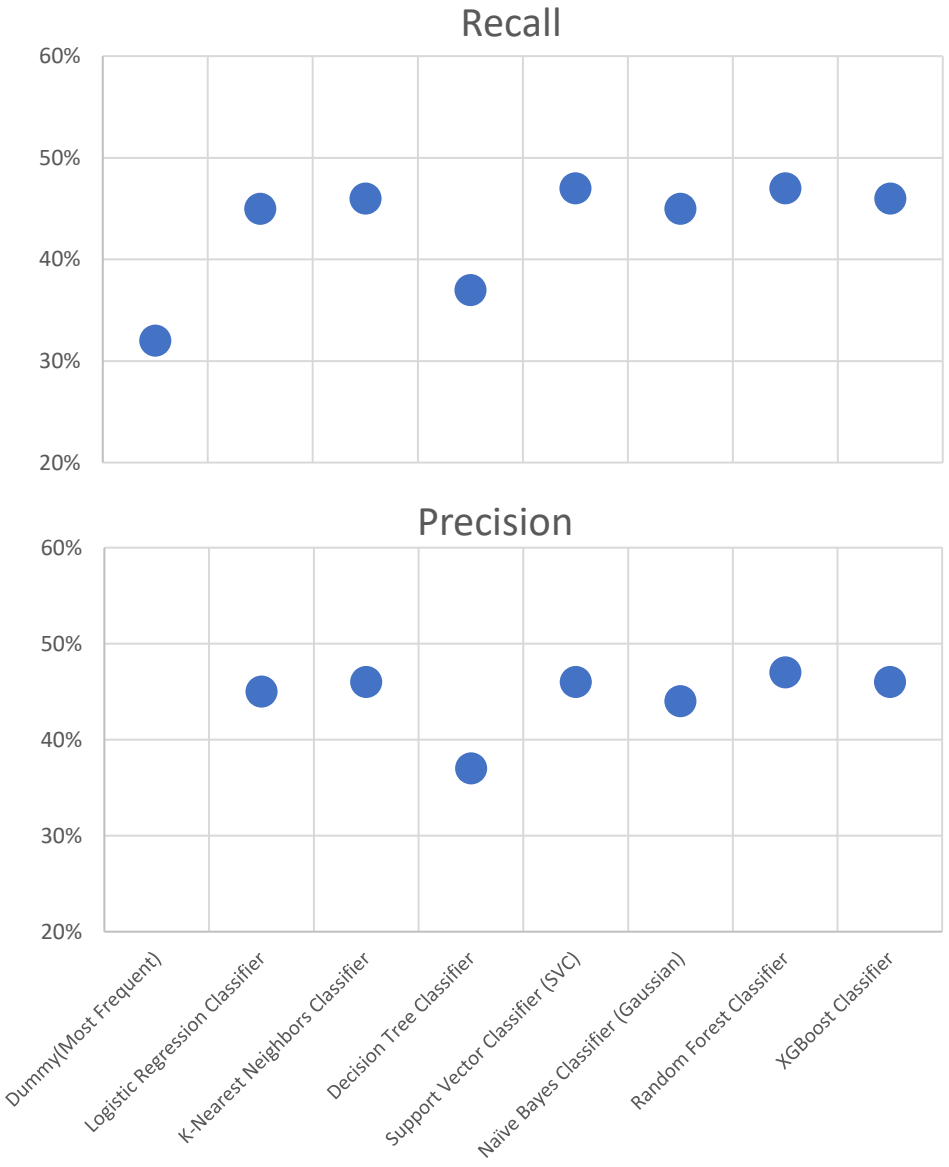
Model Name	Description
Dummy(Most Frequent)	A “baseline” model that predicts only the most frequently occurring class
Logistic Regression Classifier	Logistic regression classifier implementation that supports multi-class functionality. Per SKLearn, this uses multinomial since our data is not binary
K-Nearest Neighbors Classifier	Classifier implementing the k-nearest neighbors vote.
Decision Tree Classifier	Classifier implementing a Decision Tree using branches(feature decisions) and leaves(classes)
Support Vector Classifier (SVC)	An implementation of a Support Vector Machine algorithm that, per SKLearn documentation, uses one-vs-one scheme due to multiclass input
Naïve Bayes Classifier (Gaussian)	Classifier implementation of a Gaussian based Naïve Bayes predictor
Random Forest Classifier	A meta-estimator, that works to fit many decision trees on sub-samples of the dataset and uses averaging to improve accuracy and control overfitting
XGBoost Classifier	Similar ensemble model like Random Forest, but implements Boosting instead of Bagging

1. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

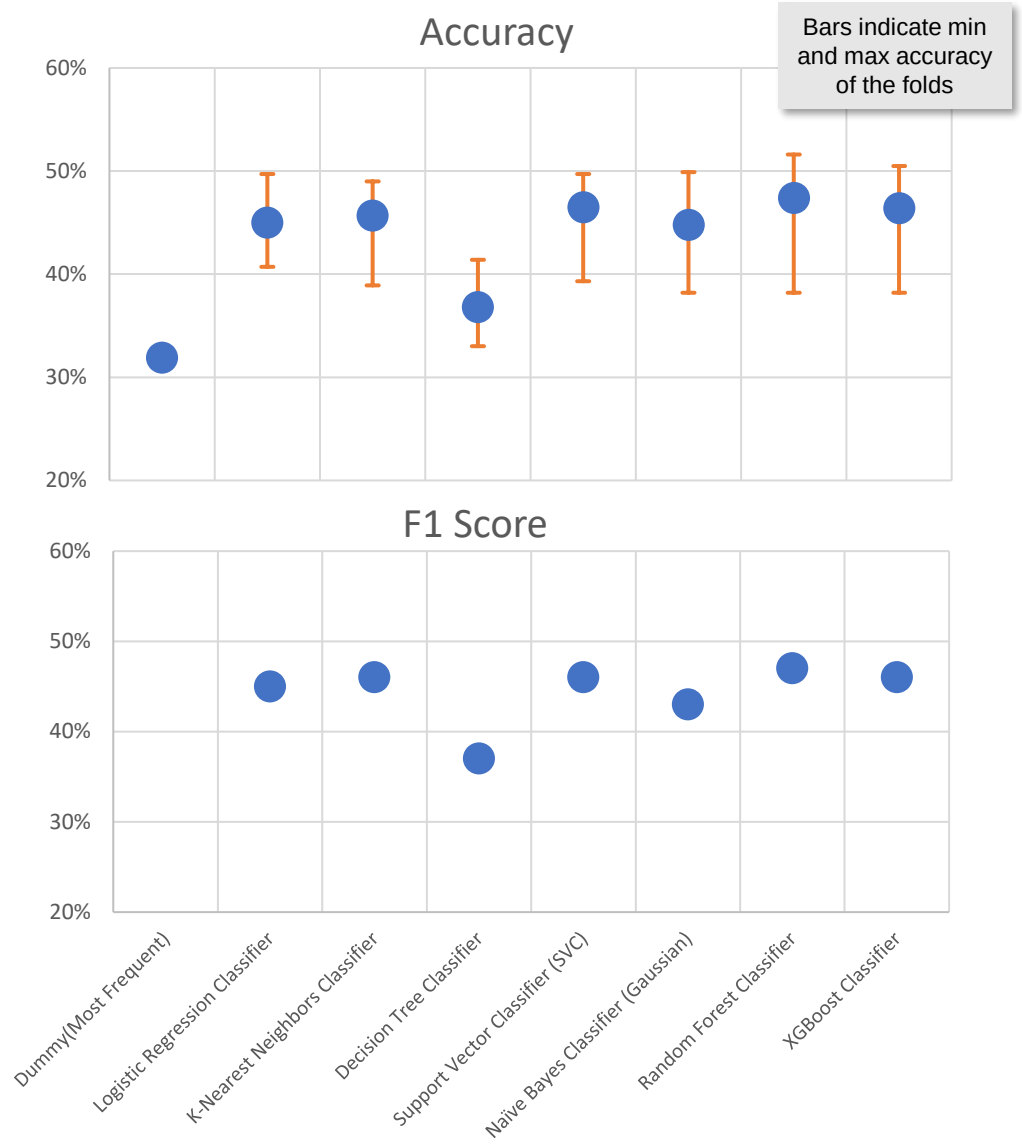
DEFAULT MODEL ANALYSIS





LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

DEFAULT MODEL ANALYSIS

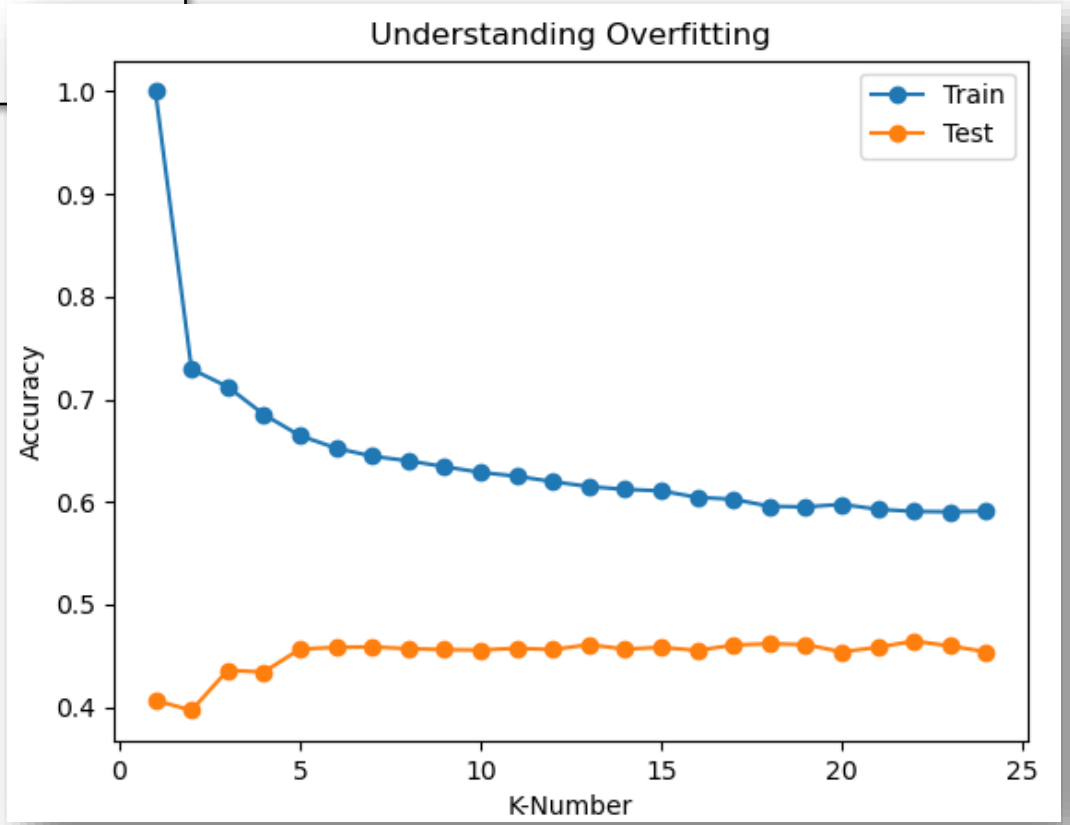


- Accuracy
  - The additional bars indicate the max and min accuracies of a single fold during validation, and the dot represents the mean accuracy
  - We can clearly see that the Decision Tree is an outlier performing poorly(not much better than our baseline Dummy classifier)
  - In general, most models are performing similar
    - The ensemble methods tend to perform slightly better and have the highest max accuracy, but also the lowest min among similar average performing methods
- F1 Score
  - Outcomes align similarly to Accuracy
  - This identifies that class distribution doesn't have a significant effect on model performance
- Best Performing Models (Further analysis to do)
  - K-Nearest Neighbors
  - Support Vector Classifier
  - Random Forest Classifier
  - XGBoost Classifier
  - These all perform at a similar level, 46-47% accuracy

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

KNN MODEL ANALYSIS – OVERFITTING ANALYSIS

- kNN models tend to overfit and underfit based off the k parameter
  - The value k relates to the error rate of the model
  - Small values can lead to overfitting and large values can lead to underfitting
  - This is n_neighbors in SKLearn



Overfitting

Overfitting is characterized by our training accuracy being significantly better than our test accuracy

Chart Analysis:

- We can clearly see the significant gap between scores with k-numbers under 5
- Once we get to a k-number of 5 our test performance levels out
- As we get to the highest k-numbers our training accuracy continues to see falloff, but we don't see significant improvement in test accuracy

Outcome:

Our optimal results are in the range of 5-10 for k-Number

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

KNN MODEL ANALYSIS – PARAMETER TUNING

- Parameter tuning is more of an art form, as there are many possibilities and combinations that can lead to an optimal(or close to optimal) solution
- To aid in the process, we look to SKLearns GridSearchCV
  - This tool aids our parameter search by looping through an input grid of parameters we provide and searching for the best combination of those
  - We use StratifiedGroupKFolds as our cross-validation method within this tool

Parameter Information:

- N_Neighbors
  - Number of neighbors to use by default kneighbors queries
- Leaf Size
  - Pass to Tree algorithm which can affect speed and memory of trees used
- P
  - Power parameter for Minkowski metric, p=1 uses Manhattan distance and p=2 uses Euclidean distance
- Weights
  - Weight functions used in prediction
- Metric
  - Distance metric used in the trees

Parameters To Test

Parameter	Unique Values
N_Neighbors	1,5,8,10,20,30
Leaf Size	1,5,8,10,20,30,40,50
P	1,2
Weights	'uniform', 'distance'
Metric	'minkowski', 'chebyshev'

Iterations Completed: 384

Best Parameter Outcomes

Parameter	Unique Values
N_Neighbors	5
Leaf Size	1
P	1
Weights	'uniform'
Metric	'minkowski'

Accuracy: 47.01%

Default Accuracy: 45%

1. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

SUPPORT VECTOR CLASSIFIER ANALYSIS– OVERFITTING ANALYSIS

- Support Vector Classifiers most import parameters tend to be C and Gamma
- We will explore overfitting when varying these parameters

Overfitting

Overfitting is characterized by our training accuracy being significantly better than our test accuracy

C Value

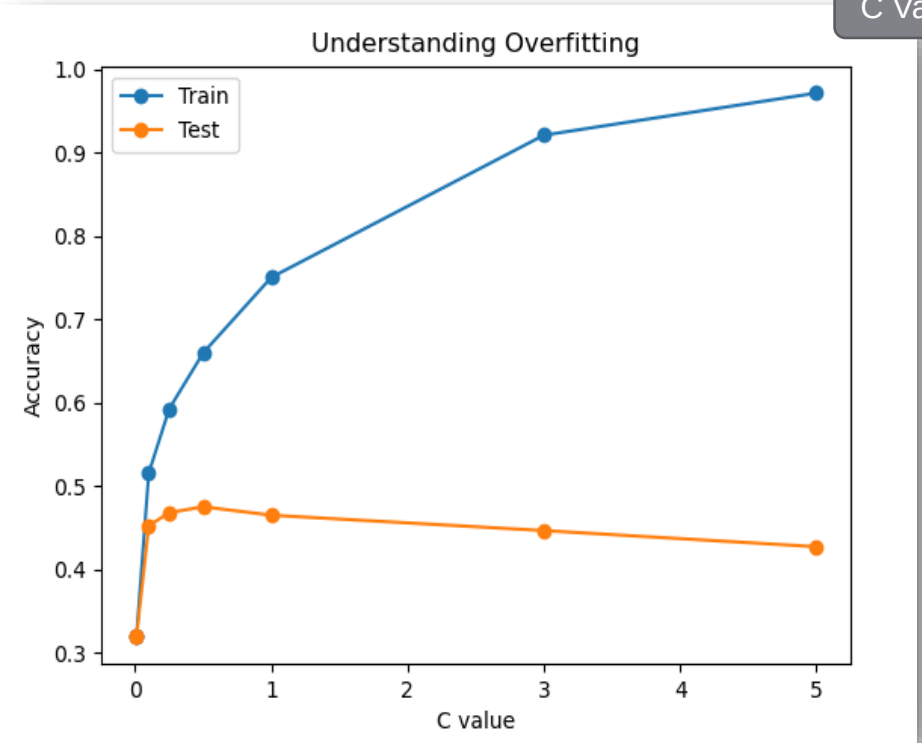


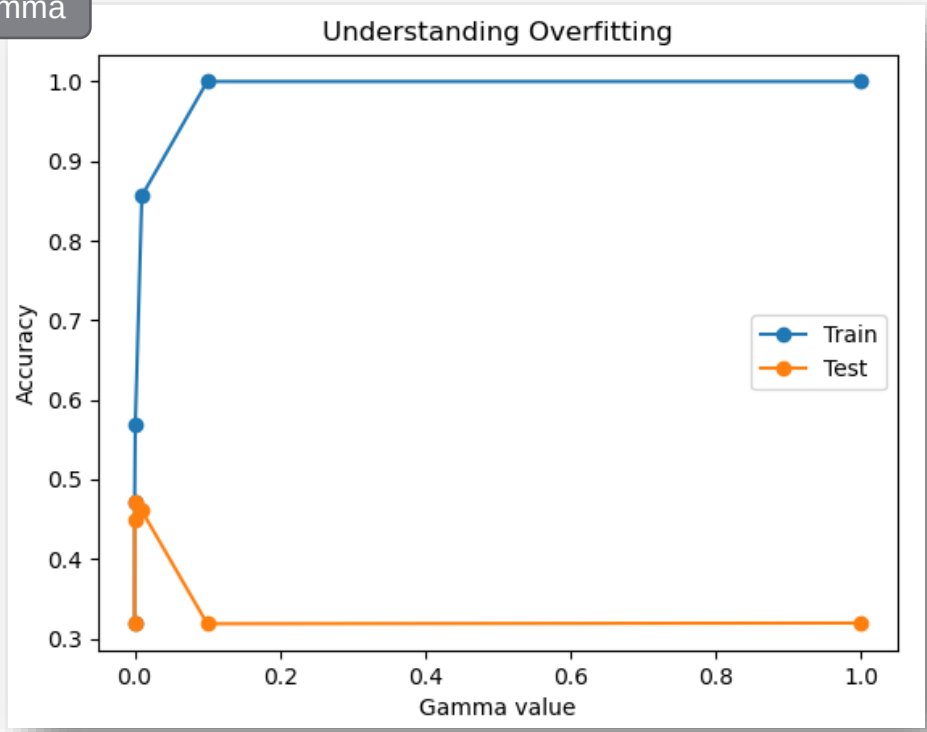
Chart Analysis:

- Gamma shows a rather uninformative chart, but suggest that we focus to very low values, close to 0
- C value chart is much more informative, as we see underfitting the closer we get to 0 and overfitting as we move greater than 1

Outcome:

Gamma should be very small, and C should most like be around 0.5

Gamma



LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

SUPPORT VECTOR CLASSIFIER ANALYSIS – PARAMETER TUNING

- Parameter tuning is more of an art form, as there are many possibilities and combinations that can lead to an optimal(or close to optimal) solution
- To aid in the process, we look to SKLearns GridSearchCV
  - This tool aids our parameter search by looping through an input grid of parameters we provide and searching for the best combination of those
  - We use StratifiedGroupKFolds as our cross-validation method within this tool

Parameter Information:

- C
  - Regularization parameter
- Gamma
  - Kernel coefficient used in the kernel algorithms
- Kernel
  - Kernel type used in the algorithm

Parameters To Test

Parameter	Unique Values
C	0.1, 1, 10, 100
Gamma	1, 0.1, 0.01, 0.001
Kernel	‘rbf’, ’poly’, ‘sigmoid’

Iterations Completed: 48

Best Parameter Outcomes

Parameter	Unique Values
C	1
Gamma	0.001
Kernel	‘rbf’

Accuracy: 47.19%

Default Accuracy: 46%

1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

RANDOM FOREST CLASSIFIER ANALYSIS– OVERFITTING ANALYSIS

- Random Forests are Decision Tree based, which tend to overfit as trees get deeper, so we will explore Max_depth variable

Overfitting

Overfitting is characterized by our training accuracy being significantly better than our test accuracy

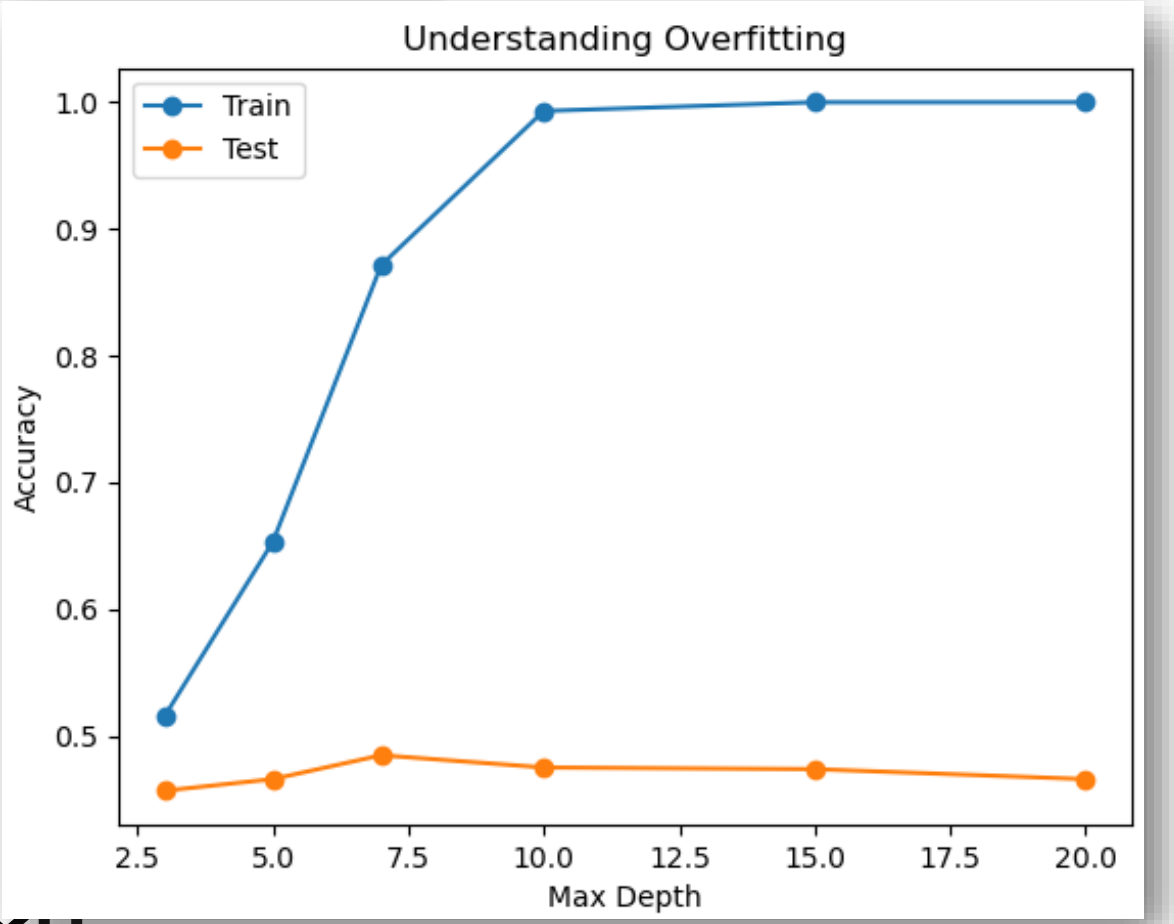


Chart Analysis:

- By the outcomes shown in our chart, a Max Depth of around 7 looks optimal given the significant bump performance on the test set
- But, looking at overfitting with Random Forests is deceptive, given how the algorithm works [1]
  - 100% Training performance is not necessarily a bad thing

Outcome:

Experiment with Max Depth between 5-10, but optimize for evaluation parameters and don't worry greatly about overfitting

1. https://towardsdatascience.com/one-common-misconception-about-random-forest-and-overfitting-47cae2e2c23b

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

RANDOM FOREST CLASSIFIER ANALYSIS – PARAMETER TUNING

- Parameter tuning is more of an art form, as there are many possibilities and combinations that can lead to an optimal(or close to optimal) solution
- To aid in the process, we look to SKLearns GridSearchCV
  - This tool aids our parameter search by looping through an input grid of parameters we provide and searching for the best combination of those
  - We use StratifiedGroupKFolds as our cross-validation method within this tool

Parameter Information:

- N_Estimators
  - Number of trees in the forest
    - Significant time factor to already lengthy model computation
- Max_Features
  - Number of features to consider when looking for the best split
- Max_Depth
  - Maximum depth of the tree
- Criterion
  - Function to measure the quality of the split

Parameters To Test

Parameter	Unique Values
N_estimators	100, 200, 500
Max_Features	‘auto’, ‘sqrt’, ‘log2’
Max_Depth	4, 5, 6, 7, 8
Criterion	‘gini’, ‘entropy’

Iterations Completed: 90

Best Parameter Outcomes

Parameter	Unique Values
N_estimators	500
Max_Features	‘auto’
Max_Depth	7
Criterion	‘gini’

Accuracy: 48.06%

Default Accuracy: 47%

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

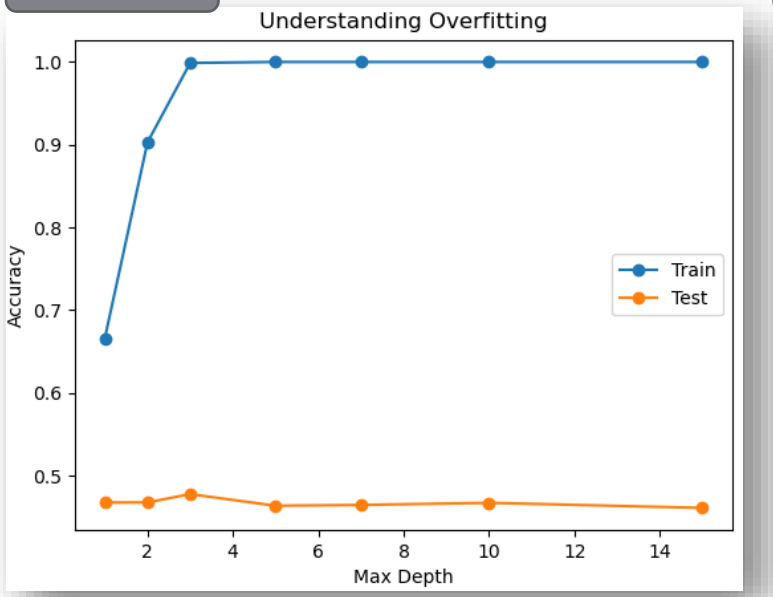
XGBOOST CLASSIFIER ANALYSIS – OVERFITTING ANALYSIS

- XGBoost is similar to Random Forests in its underlying use of Decision Trees. We will look at a couple of its important parameters

Overfitting

Overfitting is characterized by our training accuracy being significantly better than our test accuracy

Max Depth



Gamma

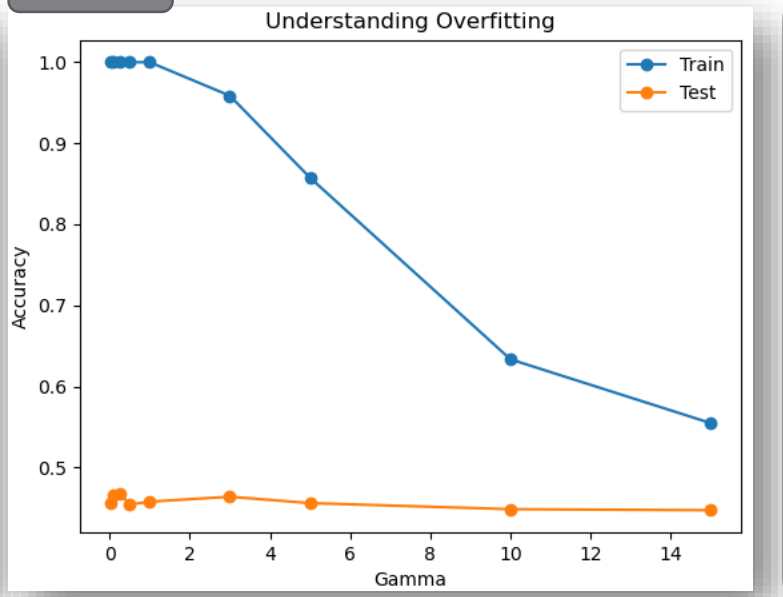


Chart Analysis:

- Given the discussion from the RandomForest overfitting analysis, our fundamental ideas of overfitting with these tree structures remains similar
- We do not see significant variations in our Test outcomes given significant variation in Training outcomes
  - ☐ We do see slight bumps in Test performance, so those may indicate the “most” optimal parameters

Outcome:

Experiment with Max Depth between 2-10 and Gamma between 0-4, but optimize for evaluation parameters and don’t worry so much about overfitting

LEARNING ALGORITHMS, OVERFITTING, AND PARAMETER VARIATION

XGBOOST CLASSIFIER ANALYSIS – PARAMETER TUNING

- Parameter tuning is more of an art form, as there are many possibilities and combinations that can lead to an optimal(or close to optimal) solution
- To aid in the process, we look to SKLearns GridSearchCV
  - This tool aids our parameter search by looping through an input grid of parameters we provide and searching for the best combination of those
  - We use StratifiedGroupKFolds as our cross-validation method within this tool

Parameter Information:

- Min_child_weight
  - Minimum sum of instance weight (hessian) needed in a child (larger = more conservative algo)
- Gamma
  - Minimum loss reduction required to make a further partition on a leaf node of the tree (larger = more conservative algo)
- Subsample
  - Subsample ratio of the training instances
- Colsample_bytree
  - subsample ratio of columns when constructing each tree
- Max_depth
  - Maximum depth of a tree (increases lead to overfitting and significant time increases

Parameters To Test

Parameter	Unique Values
Min_child_weight	1, 5, 10
Gamma	0.5, 1, 1.5, 2, 5
Subsample	0.6, 0.8, 1.0
Colsample_bytree	0.6, 0.8, 1.0
Max_depth	3, 4, 5

Iterations Completed: 405

Best Parameter Outcomes

Parameter	Unique Values
Min_child_weight	5.0
Gamma	1.0
Subsample	1.0
Colsample_bytree	1.0
Max_depth	4.0

Accuracy: 48.32%

Default Accuracy: 46%

1. https://xgboost.readthedocs.io/en/stable/parameter.html

BONUS: TRAIN AT LEAST ONE CLASSIFIER ON PREDICTING (A RELEVANT SUBSET OF) THE 9 GEMS AND 5 GEMMES FEATURES

Setup:

Model:
XGBoost

Cross Validation:
StratifiedGroupKFold

Data:
All Low & Mid Level Features

Evaluation:
Accuracy