CS 4375 Final Project Report

Tanish Paruchuri

The problem that I will be solving is whether someone is likely to get a stroke based on the features: id, gender, age, if the person has hypertension, if the person has a heart disease, if the person has ever been married, type of work, type of residence, average glucose level, body mass index (BMI), and smoking status.

This project excites me because my family has a history of stroke and it is hereditary, so learning more about the factors involved in stroke prediction could prove beneficial once I am closer to the at-risk age. Additionally, I was originally planning to be on the pre-med track. I was accepted into the University of Texas at Dallas as a Neuroscience major, so a disease that affects the brain is particularly interesting to me.

Furthermore, stroke prediction is an incredibly useful medical diagnostic tool in order to prevent serious consequences. In the United States, having a stroke is a leading cause of death and long-term disability, “someone in the United States has a stroke every 40 seconds … [someone dies of stroke] every 4 minutes.” [1]. Additionally, stroke-related costs amounted to “nearly $46 billion” [1]. Identifying patients as likely stroke candidates allows them the possibility to reduce their risk factors and circumvent this disease.

I will use the following dataset from Kaggle:
https://www.kaggle.com/fedesoriano/stroke-prediction-dataset.

The problem I am solving is a supervised learning binary classification problem. The Machine Learning models that I will use are Naive Bayes, Logistic Regression, K-Nearest Neighbors, Perceptron, Support Vector Machine, Decision Tree, Bagging (Bootstrap Aggregation) Decision Tree, Adaboost, Gradient Boosted Decision Tree, Random Forest, and Neural Network. I will compare the performance of each model and try to explain why some models are performing better than others.

Additionally, I plan to compare the performance of each model on the original feature set, polynomial features, and a subset of the original feature set that excludes any uncorrelated features; however, I may not run all for each model if I conclude that it is unnecessary. I will also use error bars in order to tune the hyper-parameters for each model if applicable.

Performance will be evaluated using accuracy and recall score. I am using accuracy as an evaluation metric since I will be balancing the dataset, thus it will be the most indicative metric of overall model performance. However, recall is the most important metric when it comes to stroke prediction itself. Recall tells us the amount of positive strokes predicted out of all positive strokes. Stroke prediction is something that would generally be used in the medical field. If the model predicts that the person will have a stroke but they aren’t actually at risk, then it is of little consequence. However, if the model does not predict that the person will have a stroke but they actually are at risk, there are substantial consequences as that person won’t know to proactively reduce their risk factors. I decided not to use F1 or ROC AUC as the dataset won’t be imbalanced, they aren’t as indicative of model performance as accuracy, and they aren’t as meaningful as recall.

Lastly, I'm seeking a model with a recall score of at least 70%, but ideally between 80% and 90% since "anything between 70%-90% is … realistic … [and] consistent with industry standards." [2]

Exploratory Data Analysis:

First, I checked how the dataset is formatted.

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Next, I checked if there were any missing values or NANs in the dataset.

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

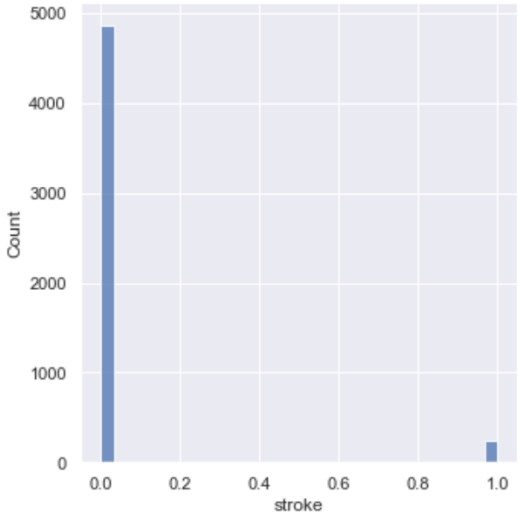
bmi has 201 NAN values, so I replaced them with the mean of the rest of the bmi values.

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

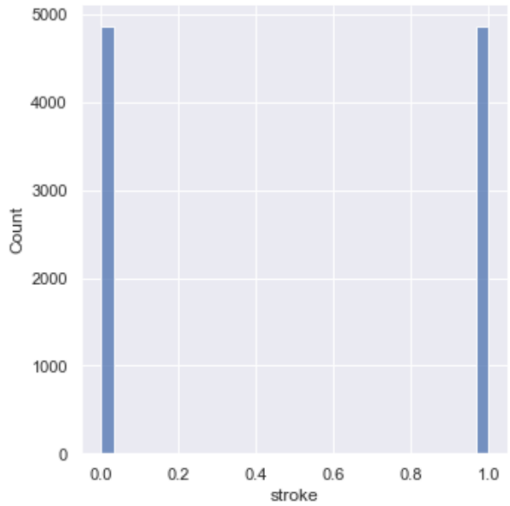
Next, I replaced the categorical data such as gender and ever_married with numbers in order to run the models.

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	0	67.0	0	1	1	3	1	228.69	36.600000	0	1
1	51676	1	61.0	0	0	1	4	0	202.21	28.893237	1	1
2	31112	0	80.0	0	1	1	3	0	105.92	32.500000	1	1
3	60182	1	49.0	0	0	1	3	1	171.23	34.400000	2	1
4	1665	1	79.0	1	0	1	4	0	174.12	24.000000	1	1

Next, I plotted the number of samples per class of the target variable stroke.

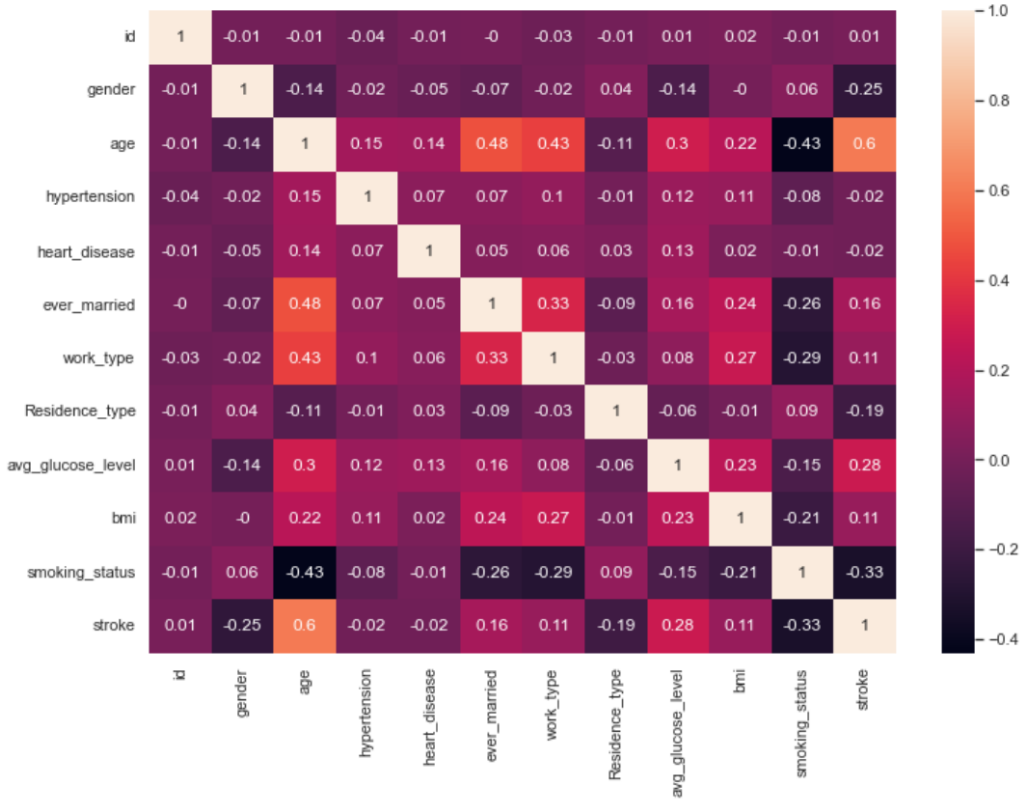


There are almost 4800 samples per class 0 and about 200 samples per class 1. The dataset is extremely imbalanced. Due to this, I will need to do some resampling in order to balance the dataset (make the number of samples of each class equivalent). I decide to use oversampling which will generate samples for the minority class so that it matches the number of samples in the majority class. For this, I use SMOTE, Synthetic Minority Over-sampling Technique, which “[increases] the number of cases in your dataset in a balanced way … by generating new instances from existing minority cases that you supply as input.” [3]



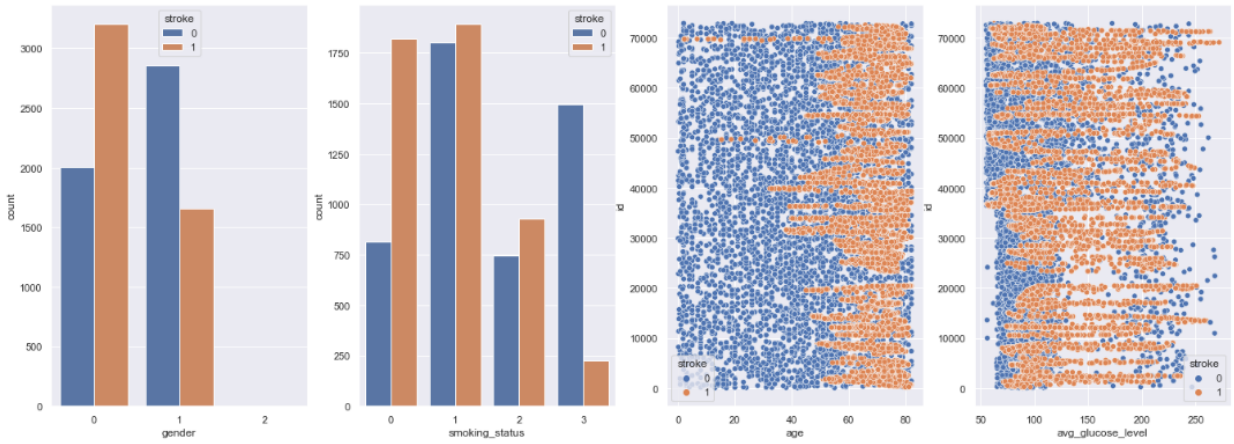
The number of samples in the 1 class now match the number of samples in the 0 class, so the dataset is balanced.

Finally, I was able to compute the correlation matrix to determine the correlation between each feature and every other feature and the target.



I also decided to plot some graphs of the features with the highest correlation scores: gender, smoking_status, age, and avg_glucose_level. I did this to try and get an idea of the degree of the hypothesis function (if it could benefit from polynomial features). Note: I plotted age vs id as id is uncorrelated with all features and it gives the graph some dimensionality.

Based on the last two graphs, along with the fact that all but age are weakly correlated, I'm guessing that the hypothesis function should benefit from polynomial features because the density of the positive stroke class on the highest correlated feature, age, is very weak in the lowers ranges before it becomes extremely dense in the higher ranges only. None of the features are highly correlated with each other, so there doesn't seem to be any redundant features. However, most of the features are weakly correlated with the target. id, hypertension, and heart_disease in particular seem to have next to no correlation with the target.


stroke
0
1
count
gender
smoking_status
id
age
avg_glucose_level

Model Analysis:

For each model, I ran the original set of data, the original dataset with 2-degree polynomial features, and a subset of features. Each set of data was split into a 70% training and 30% test set. I also normalized all the features using sklearn.preprocessing.StandardScaler since each of the features are on incomparable scales. Also because "some machine learning algorithms benefit from normalization … such as k-nearest neighbors and artificial neural networks" and algorithms that utilize "gradient descents can converge faster." [4]

I chose to obtain the training and test accuracy to study overfitting/underfitting and the test recall to be the metric used to compare model performance. Here are the training accuracy, test accuracy, and test recall results for each model in the following order: the original set of data, the original dataset with 2-degree polynomial features, and a subset of features.

Gaussian Naive Bayes:

I chose Gaussian Naive Bayes since the features are a mix of both discrete and continuous variables.

```
Gaussian Naive Bayes Training Accuracy: 0.8005878030859662
Gaussian Naive Bayes Test Accuracy: 0.8141926636955776
Gaussian Naive Bayes Test Recall score: 0.884404924760602

Gaussian Naive Bayes Training Accuracy: 0.6919911829537105
Gaussian Naive Bayes Test Accuracy: 0.7072334590332533
Gaussian Naive Bayes Test Recall score: 0.9001367989056087

Gaussian Naive Bayes Training Accuracy: 0.8007347538574577
Gaussian Naive Bayes Test Accuracy: 0.8138498457319163
Gaussian Naive Bayes Test Recall score: 0.8837209302325582
```

The model does not seem to be underfitting or overfitting as both the train and test accuracies are similar and they are decently high.

Logistic Regression:

```
Logistic Regression Training Accuracy: 0.8255694342395298
Logistic Regression Test Accuracy: 0.8337332876242715
Logistic Regression Test Recall score: 0.8481532147742818

Logistic Regression Training Accuracy: 0.858192505510654
Logistic Regression Test Accuracy: 0.85978745286253
Logistic Regression Test Recall score: 0.8816689466484268

Logistic Regression Training Accuracy: 0.8245407788390889
Logistic Regression Test Accuracy: 0.8357901954062393
Logistic Regression Test Recall score: 0.8481532147742818
```

The model does not seem to be underfitting or overfitting as both the train and test accuracies are similar and they are decently high.

K-Nearest Neighbors:

I plotted the Test Recall vs K for values of K from 1 to 20, in order to determine the ideal value of K.

Based on this graph I chose K = 1, thus performing 1-Nearest Neighbor.

```
1-Nearest Neighbor Training Accuracy: 1.0
1-Nearest Neighbor Test Accuracy: 0.9115529653753857
1-Nearest Neighbor Test Recall score: 0.9651162790697675

1-Nearest Neighbor Training Accuracy: 1.0
1-Nearest Neighbor Test Accuracy: 0.9098388755570792
1-Nearest Neighbor Test Recall score: 0.9637482900136799

1-Nearest Neighbor Training Accuracy: 1.0
1-Nearest Neighbor Test Accuracy: 0.8899554336647241
1-Nearest Neighbor Test Recall score: 0.9302325581395349
```

The model is definitely overfitting since the training set has a 100% accuracy. This makes sense because in 1-Nearest Neighbor the training points are precisely being predicted on themselves since the training points are the model. However, the test accuracies are high, so this is a good model.

Perceptron:

Similarly to K-Nearest Neighbors, I plotted Test Recall vs n_iter_no_change to find the ideal value for n_iter_no_change.

Based on this graph I chose n_iter_no_change = 8.
I did the same for 2-Degree Polynomial features.

Based on this graph I chose n_iter_no_change = 51.

```
Perceptron Training Accuracy: 0.7753122703894195
Perceptron Test Accuracy: 0.7908810421666095
Perceptron Test Recall score: 0.9288645690834473

Perceptron Training Accuracy: 0.8055841293166789
Perceptron Test Accuracy: 0.8179636612958519
Perceptron Test Recall score: 0.939124487004104

Perceptron Training Accuracy: 0.7259368111682586
Perceptron Test Accuracy: 0.7394583476174151
Perceptron Test Recall score: 0.7428180574555403
```

The model does not seem to be underfitting or overfitting as both the train and test accuracies are similar and they are decently high.

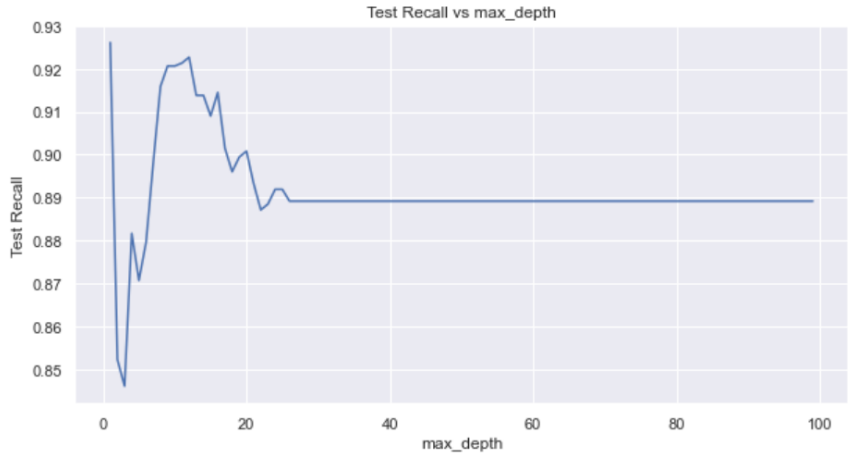
Support Vector Machines:

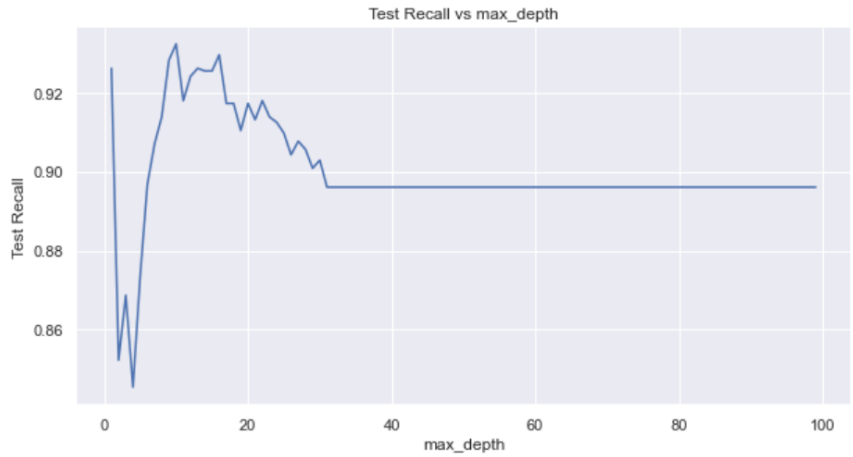
```
Support Vector Classification Training Accuracy: 0.8855253490080823
Support Vector Classification Test Accuracy: 0.8669866300994172
Support Vector Classification Test Recall score: 0.960328317373461

Support Vector Classification Training Accuracy: 0.7134459955914768
Support Vector Classification Test Accuracy: 0.6931779225231403
Support Vector Classification Test Recall score: 0.9863201094391245

Support Vector Classification Training Accuracy: 0.8689199118295371
Support Vector Classification Test Accuracy: 0.8519026396983201
Support Vector Classification Test Recall score: 0.9514363885088919
```

The model does not seem to be underfitting or overfitting as both the train and test accuracies are similar and they are decently high.

Decision Tree:

Again, I plotted Test Recall vs max_depth to find the ideal value for max_depth for all and 2-Degree polynomial features.





Based on this graph I got max_depth = 1 for original features and max_depth = 13 for 2-Degree polynomial features.

```
Decision Tree Training Accuracy: 0.7844232182218956
Decision Tree Test Accuracy: 0.8021940349674322
Decision Tree Test Recall score: 0.9261285909712722

Decision Tree Training Accuracy: 0.9672299779573843
Decision Tree Test Accuracy: 0.8961261570106274
Decision Tree Test Recall score: 0.9261285909712722

Decision Tree Training Accuracy: 1.0
Decision Tree Test Accuracy: 0.8700719917723688
Decision Tree Test Recall score: 0.8693570451436389
```

The original features do not seem to be underfitting or overfitting as both the train and test accuracies are similar and they are decently high. The 2-degree polynomial features seems to be overfitting, but again the test accuracies are very high as well so it's a good model.

Bagging (Bootstrap Aggregation) Decision Tree:

```
Bagging Decision Tree Training Accuracy: 0.995444526083762
Bagging Decision Tree Test Accuracy: 0.9259513198491601
Bagging Decision Tree Test Recall score: 0.9357045143638851

Bagging Decision Tree Training Accuracy: 0.9963262307127112
Bagging Decision Tree Test Accuracy: 0.9208090503942407
Bagging Decision Tree Test Recall score: 0.9357045143638851

Bagging Decision Tree Training Accuracy: 0.995444526083762
Bagging Decision Tree Test Accuracy: 0.901611244429208
Bagging Decision Tree Test Recall score: 0.8987688098495212
```

The model seems to be slightly overfitting as both the train accuracies are both slightly greater than their respective test accuracies. This model is still good as the test accuracies are high.

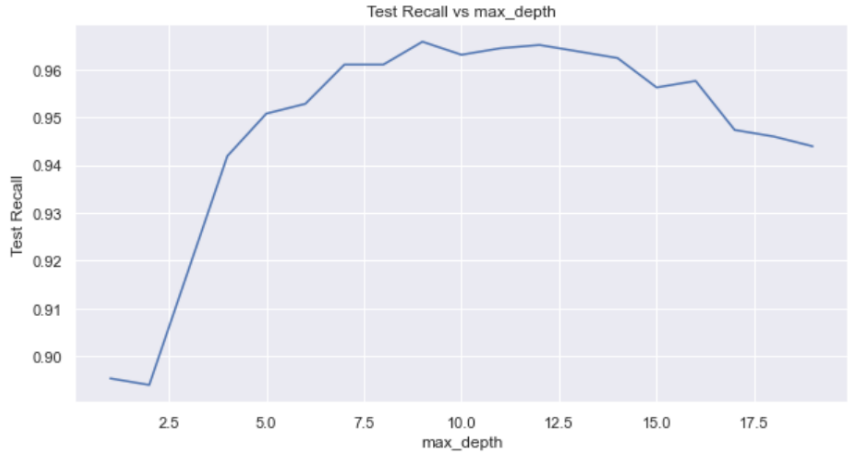
Adaboost:

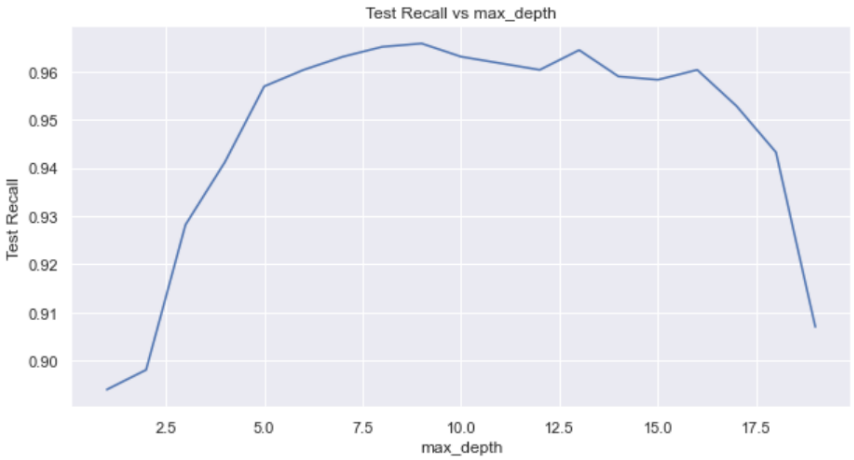
```
Adaboost Training Accuracy: 1.0
Adaboost Test Accuracy: 0.8817278025368529
Adaboost Test Recall score: 0.8926128590971272

Adaboost Training Accuracy: 1.0
Adaboost Test Accuracy: 0.8865272540281111
Adaboost Test Recall score: 0.9008207934336525

Adaboost Training Accuracy: 1.0
Adaboost Test Accuracy: 0.8666438121357559
Adaboost Test Recall score: 0.8666210670314638
```

The model is definitely overfitting since the training set has a 100% accuracy. However, the test accuracies are high, so this is a good model.

Gradient Boosted Decision Trees:

Again, I plotted Test Recall vs max_depth to find the ideal value for max_depth for all and 2-Degree polynomial features.





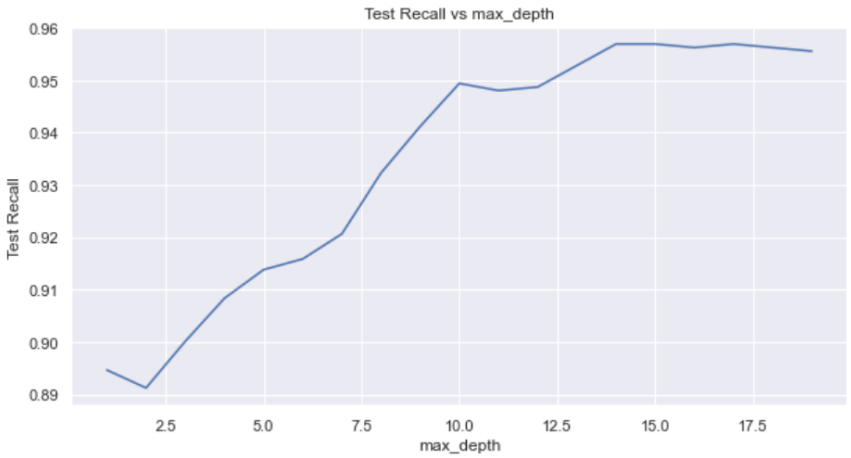
Based on these graphs I got max_depth = 9 for original features and max_depth = 9 for 2-Degree Polynomial features.

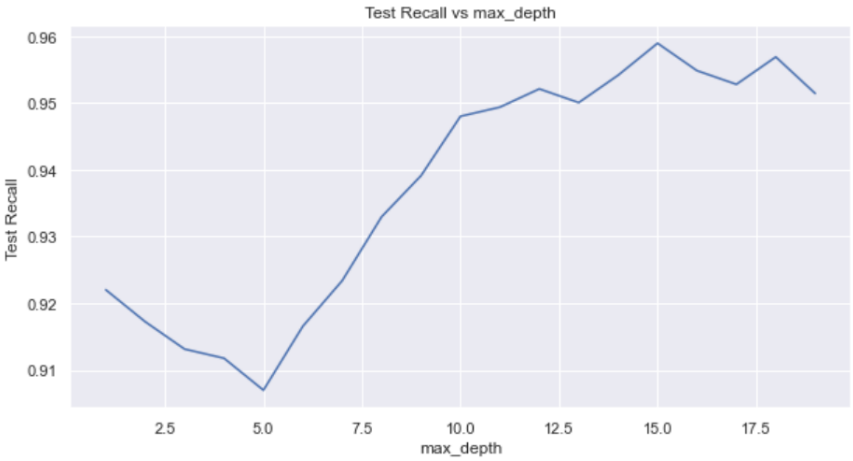
```
Gradient Boosted Decision Trees Training Accuracy: 1.0
Gradient Boosted Decision Trees Test Accuracy: 0.9544052108330476
Gradient Boosted Decision Trees Test Recall score: 0.9658002735978112

Gradient Boosted Decision Trees Training Accuracy: 1.0
Gradient Boosted Decision Trees Test Accuracy: 0.951319849160096
Gradient Boosted Decision Trees Test Recall score: 0.9658002735978112

Gradient Boosted Decision Trees Training Accuracy: 0.8890521675238795
Gradient Boosted Decision Trees Test Accuracy: 0.8810421666095304
Gradient Boosted Decision Trees Test Recall score: 0.9015047879616963
```

The model is definitely overfitting since the training set has a 100% accuracy. However, the test accuracy is extremely high, so this is a great model.

Random Forest:

Again, I plotted Test Recall vs max_depth to find the ideal value for max_depth for all and 2-Degree polynomial features.





Based on these graphs I got max_depth = 15 for original features and max_depth = 15 for 2-Degree Polynomial features.

```
Random Forest Training Accuracy: 0.9875091844232182
Random Forest Test Accuracy: 0.9321220431950634
Random Forest Test Recall score: 0.9569083447332422

Random Forest Training Accuracy: 0.9866274797942689
Random Forest Test Accuracy: 0.9331504970860474
Random Forest Test Recall score: 0.9589603283173734

Random Forest Training Accuracy: 1.0
Random Forest Test Accuracy: 0.917723688721289
Random Forest Test Recall score: 0.9295485636114911
```

The model seems to be slightly overfitting as both the train accuracies are both slightly greater than their respective test accuracies. This model is still great as the test accuracies are extremely high.

Neural Network (Multi-Layer Perceptron):

I chose to use a Multi-Layer Perceptron as it works great for classification and its capability to learn non-linear models.

```
Neural Network Training Accuracy: 0.9362233651726671
Neural Network Test Accuracy: 0.9115529653753857
Neural Network Test Recall score: 0.9377564979480164
```

```
Neural Network Training Accuracy: 0.9801616458486407
Neural Network Test Accuracy: 0.9170380527939664
Neural Network Test Recall score: 0.9418604651162791

Neural Network Training Accuracy: 0.9108008817046289
Neural Network Test Accuracy: 0.8923551594103531
Neural Network Test Recall score: 0.9186046511627907
```

The original features do not seem to be underfitting or overfitting as both the train and test accuracies are similar and they are decently high. The 2-degree polynomial features seems to be overfitting, but again the test accuracies are very high as well so it's a great model.

All of these models were run with tuned hyperparameters and I did increase the maximum iterations for Logistic Regression and Neural Network since they weren't able to converge with the default value. I also added a random state to Decision Tree, Bagging Decision Tree, Adaboost, Gradient Boosted Decision Tree, Random Forest, and Neural Network for model performance consistency.

Recall that in the data analysis portion, I hypothesized that the models would benefit from polynomial features since the density of the positive stroke class significantly increases around the age 50 mark. The negative stroke class is spread out in all age ranges while almost all of the positive stroke class is in the 50+ range.

Observations from using 2-Degree Polynomial features on each model:

- Gaussian Naive Bayes
  Lowered Train and Test Accuracy by about 10% each. Insignificantly Greater Recall.

- Logistic Regression
  Insignificantly Greater Train and Test Accuracy. Greater Recall by about 4%.

- 1-Nearest Neighbor
  Same Train Accuracy. Insignificantly Lowered Test Accuracy. Insignificantly Lowered Recall.

- Perceptron
  Greater Train and Test Accuracy by about 3%. Lowered Recall by about 4%.

- Support Vector Machine
  Insignificantly Greater Train and Test Accuracy. Insignificantly Greater Recall.

- Decision Tree
  Insignificantly Same Train Accuracy. Insignificantly Greater Test Accuracy. Insignificantly Greater Recall.

- Bagging Decision Tree
  Insignificantly Greater Train Accuracy. Insignificantly Lowered Test Accuracy. Same Recall.

- Adaboost
  Insignificantly Same Train Accuracy. Insignificantly Greater Test Accuracy. Insignificantly Greater Recall.

- Gradient Boosted Decision Tree
  Insignificantly Greater Train and Test Accuracy. Insignificantly Greater Recall.

- Random Forest
  Insignificantly Same Train Accuracy. Insignificantly Lowered Test Accuracy. Insignificantly Lowered Recall.

- Neural Network
  Greater Train Accuracy by about 5%. Insignificantly Greater Test Accuracy. Insignificantly Greater Recall.

I only specifically marked any changes if they were 3% or more. If it was noted as insignificant, then the changes were less than 3%. I did try using 3-Degree polynomial features as well and the results were extremely similar to 2-Degree. Based on these results, it doesn't seem like adding polynomial features was very beneficial. Now to explain some of the results. Polynomial features on Naive Bayes can skew the low conditional probabilities to be even lower, so that's probably why the accuracy dropped.

It doesn't seem like adding polynomial features was greatly beneficial for any of the models, but it does give a slight edge over just the original features in recall score for most of the models. Furthermore, it does overfit for Random Forest and Neural Network, but the recall score is the same or slightly increases compared to the original features. Therefore, I will utilize 2-degree polynomial features for my final phase of model analysis.

In the final phase of model analysis, I will randomize the samples in the training and test split and obtain the test recall values over 20 runs to perform cross-validation. I will then compute the mean and standard deviation for each model and plot the error bars in order to find the optimal model for stroke prediction. I will first reduce the number of models by only using the models with a test recall score of at least 0.95, which I chose arbitrarily after looking at the test recall scores of all the models. Note: I am using 2-degree polynomial features, so I am looking at the 2-degree polynomial features test recall for each model. This leaves me with 1-Nearest Neighbor, Support Vector Machines, Gradient Boosted Decision Trees, and Random Forest.

Results:

As suggested in my project proposal, I tried many different subsets of features to see if it improved performance.

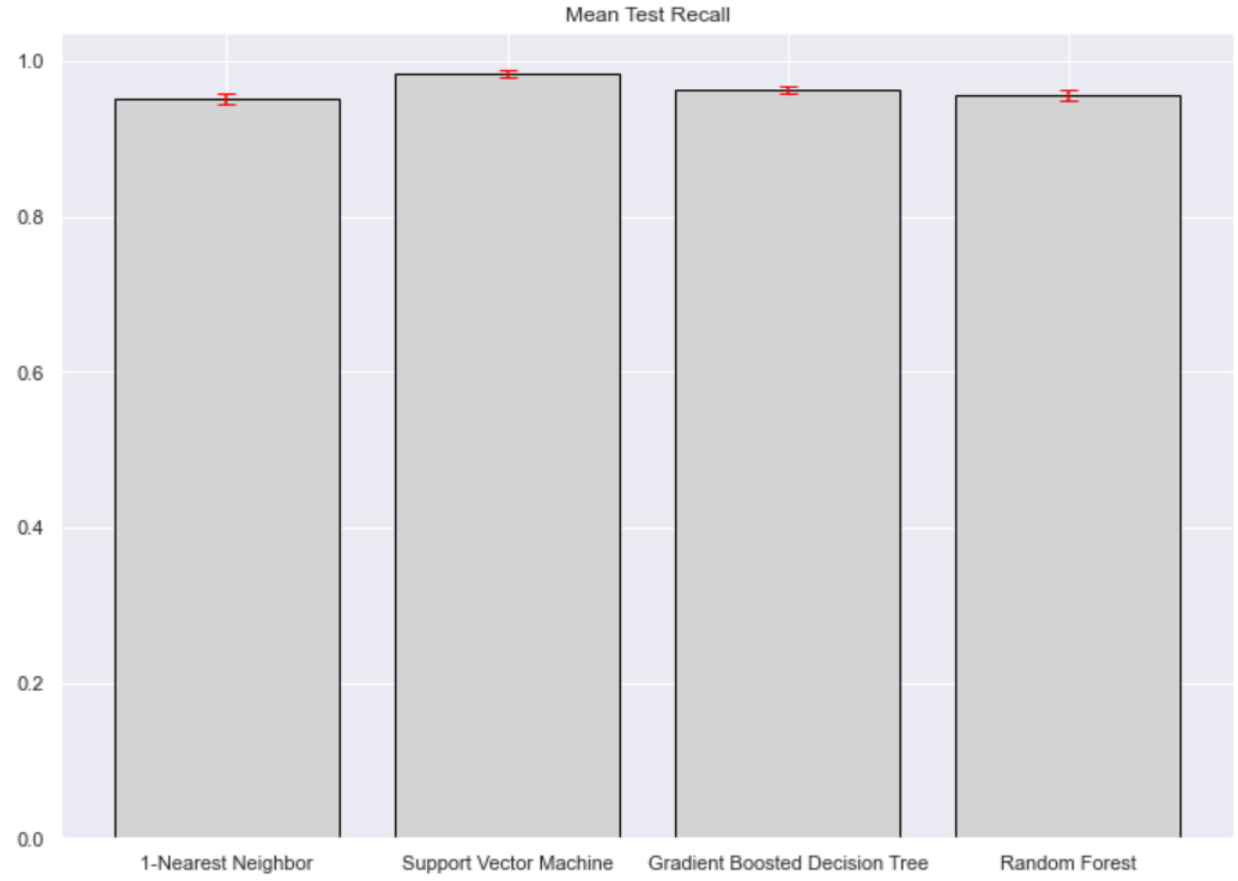
1. Removing just id since its uncorrelated to all features and the target
2. Removing id, ever_married, work type since the last 2 are mediumly correlated with each other (thought it might be redundant)
3. Removing id, hypertension, heart_disease since the last 2 are uncorrelated with the target (thought it might be irrelevant)
4. Removing id, ever_married, work_type, hypertension, heart_disease (combined last two).

It turns out that using a subset of features performs worse for all models than using all the features. This is very surprising especially since removing just id, which is uncorrelated with every single other feature and the target, decreases model performance. According to The Stanford AI Lab Blog, there are "two natural explanations" as to why "removing spurious features can lead to drops in accuracy." [5]

1. Core (non-spurious) features can be noisy or not expressive enough so that even an optimal model has to use spurious features to achieve the best accuracy. [5]
2. Removing spurious features can corrupt the core features. [5]

This makes sense as age is the only feature that is decently correlated with the target. All the other features are only weakly correlated with the target, so they must not be expressive enough to achieve the best accuracy without an uncorrelated feature such as id.

```
1-Nearest Neighbor. Mean = 0.9519493175185956. Standard Deviation = 0.0064865732403053735.
Support Vector Machine. Mean = 0.9840450983343529. Standard Deviation = 0.004091868643934485.
Gradient Boosted Decision Tree. Mean = 0.9634238269394959. Standard Deviation = 0.0047100398567676725.
Random Forest. Mean = 0.956663922263789. Standard Deviation = 0.0070479492449923775.
```



Based on these results, that were obtained using the procedure described at the end of the model analysis section, we can see that Support Vector Machines is the most optimal model for stroke prediction using these features. This is evident, since it has the highest mean and the lowest standard deviation test recall score.

Lastly, for the task specific metric, I have obtained the following results using the mean test recall of the most optimal mode (Support Vector Machines):

```
On average, 4783 actual strokes were correctly predicted.
On average, 78 actual strokes were incorrectly predicted.
```

The obtained optimal model performance of 0.9840450983343529 far surpasses my initial expectations. This means that the model is excellent, almost perfect, which I thought may be cause for concern. My presumption was that this is the result of none of the features being highly correlated with the target - one feature was mediumly correlated and the rest were weakly correlated. I originally thought that I was overfitting on the test set via hyperparameter optimization, but the same hyper parameters ran on many randomized test splits led to very small standard deviation value, so all the calculated means are very close to each other regardless of the test set.

I then decided to compute the mean and standard deviation of the test accuracy using the same procedure that was used for test recall in order to investigate this.

```
Support Vector Machine Test Accuracy.
Mean = 0.6870929036681522. Standard Deviation = 0.007274589918951101.
```

It turns out that the model is just decent in terms of overall model performance in being predictive for both test classes. Of course, we could have definitely obtained a model with a much higher mean test accuracy, which is evident by the much higher test accuracy scores in the initial model evaluation phase. Thus, I concluded that nothing unforeseen was occurring (major forms of bias that would skew the results). Furthermore, I do believe that the model's high performance is a result of the non-robust dataset itself. Having only one standout indicative feature makes it significantly easier for the model to predict the positive class.

References

[1] “Stroke facts,” Centers for Disease Control and Prevention, May 25, 2021. [Online]. Available: https://www.cdc.gov/stroke/facts.htm#:~:text=Stroke%20Statistics&text=Every%204%20minutes%2C%20someone%20dies%20of%20stroke.&text=Every%20year%2C%20more%20than%20795%2C000,are%20first%20or%20new%20strokes.&text=About%20185%2C000%20strokes%E2%80%94nearly%201,have%20had%20a%20previous%20stroke. [Accessed March 09, 2022].

[2] K. Barkved, “How To Know if Your Machine Learning Model Has Good Performance,” *obviously.ai*. [Online]. Available: https://www.obviously.ai/post/machine-learning-model-performance#:~:text=Good%20accuracy%20in%20machine%20learning,also%20consistent%20with%20industry%20standards. [Accessed: May 04, 2022].

[3] Likebupt, “SMOTE,” *Azure Machine Learning* | *Microsoft Docs*, 11-Apr-2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/component-reference/smote#:~:text=Synthetic%20Minority%20Oversampling%20Technique%20(SMOTE,that%20you%20supply%20as%20input. [Accessed: May 04, 2022].

[4] “Normalization in Machine Learning,” *deepchecks*, 05-Aug-2021. [Online]. Available: https://deepchecks.com/glossary/normalization-in-machine-learning/. [Accessed: May 06, 2022].

[5] F. Khani, “Removing Spurious Features can Hurt Accuracy and Affect Groups Disproportionately,” *The Stanford AI Lab Blog*, 24-Jan-2021. [Online]. Available: https://ai.stanford.edu/blog/removing-spuriousfeature/. [Accessed: May 06, 2022].