Diabetes Dataset Exploration & Prediction

CSI 5155 Final Project

Nicolas Fleece
S# 8327287

March 26, 2021

Abstract

This report details the study of the diabetes dataset, which is analyzed in the paper *"Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records"*. Through the analysis of this data, we can use machine learning algorithms in order to be able to predict whether the patient will be readmitted to the hospital or not. This may give insights into what factors affect readmission. In addition, we explore the prediction of gender based on the same data. This exploration may reveal whether or not there are certain risk factors for the different genders. Lastly, we use semi-supervised learning where we treat some of the training data as unlabelled and use different techniques to train on this unlabelled data. Overall, I found that I was able to achieve good quality predictions on all three of the tasks that are present in this report, and propose some new directions that may be possible with assistance from domain experts.

Contents

1 Introduction

1.1 Supervised Learning Tasks

We explore two different supervised learning problems that use all of the labels provided in the dataset. We specifically explore two different problems, namely predicting whether or not a patient will be readmitted, and trying to determine what gender someone is based on their medical information in the dataset.

1.1.1 Readmitted Prediction

The first problem is predicting whether or not someone will be readmitted based on the features that are provided in the dataset. The motivation behind this problem is fairly straightforward, we want to develop a model that can predict if someone is going to need to be readmitted in the future, so that it can be planned for and the appropriate adjustments can be made. In addition, exploration of the dataset and development of these models may give us hints as to whether or not there are features that carry significant weight in determining if an individual will have to be readmitted in the future. For this problem the positive class is defined as a patient who was readmitted, and the negative class is defined as someone who was not readmitted. It is also important to note that this problem was transformed into a binary classification problem as I felt it represented the problem better.

1.1.2 Gender Prediction

The reasoning behind the problem of predicting what gender a patient is is a little more complex. The logic I had behind this is to try and determine if there are different risk factors that may cause a person of a specified gender to become hospitalized. We could also potentially determine whether or not a particular gender has any specific risk factors that may lead them to receive medication of a specific type, or be treated by a specific kind of medical professional. This problem is a binary classification problem where male is the positive class and female is the negative class.

1.2 Semi-Supervised Learning Task

The semi-supervised learning task is the same as our supervised problem of predicting whether or not a patient will be readmitted, except we remove some of the training target variables. This leaves us with two different sets of training values; one set is normal and has the associated labels, and the other set contains only the training features, but not associated labels. Using three different algorithms, we can change this ratio of labelled/unlabelled data and observe the results.

2 Algorithms

2.1 Supervised Learning

For supervised learning approaches, I ensured to utilize different algorithms from all of the sections covered in class, namely trees, neural networks, distance-based algorithms, bayesian approaches and ensembles. Note that for all of these supervised methods I utilized scikit-learn. Any hyperparameters that are not specified in this paper can be assumed to be the default as defined in the scikit-learn documentation.

2.1.1 Decision Tree

Decision Tree is the simple tree-based classifier that I chose to use. The primary concern with this type of classifier is which feature to split on at any particular branch. In my case, I use the gini index, the calculation for this index is defined in equation 1. In this equation $\dot{p}$ is defined as the probability of a given value of a feature being a part of the positive class, and $1 - \dot{p}$ being the inverse, where this is the probability of the negative class. We choose the feature with the lowest gini index value to split on.

$$2\dot{p}(1 - \dot{p}) \tag{1}$$

2.1.2 Random Forest

Random forest is an ensemble variation on decision trees. In this case, we take a random subset of the features with replacement, and construct a decision tree based on this subset of the data. Again, identical to decision trees, we use the gini index value as defined in equation 1. We use 100 of these trees using subsets of data, which increases the accuracy by forcing the trees to learn on different parts of the dataset. When we use the random forest classifier to classify a validation example, we simply take all of these smaller decision trees, and have them vote on the target class, taking the majority class as the prediction.

2.1.3 Multi-Layer Perceptron

Multi-layer perceptron is a neural network design where we have many connected neurons and utilizing gradient descent through a loss function, we can adjust the weights on the inputs of these neurons to minimize the loss. The key hyperparameter for this model is the size of the hidden layer. I defined this as the default which is a single hidden layer of size 100. I experimented with some other hidden layer sizes, and multiple hidden layers, and will describe my decision more in the results section of this report.

2.1.4 K Nearest Neighbours

K Nearest Neighbour is the distance-based classifier that I chose to use. This classifier simply classifies based on the K nearest samples and uses the majority rule to determine the target class. The hyperparameter that is the most important in this case is how many of the K nearest samples to check; in the case of this report, I use the default which is $K = 5$.

2.1.5 Naive Bayes

For the baysian learner, I chose to use a simple naive bayes implementation. Specifically in scikit-learn I utilized the Gaussian Naive Bayes class, which is a slightly more complex variant. The core of bayesian learners is determining the target class based on probabilities. It uses the probabilities of the target class based on the values of each of the features, and using equation 2, we can determine the target class based on the highest probability. This equation is defined within the scikit-learn documentation.

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{2}$$

2.2 Semi-Supervised Learning

2.2.1 Label Propagation & Label Spreading

Our first two semi-supervised learning methods are rather similar, and both are implemented via scikit learn. Both of these methods work similarly in that they construct a similarity graph over the entire dataset, and utilize this similarity graph in order to label the unlabelled training data. The difference between the two is that Label Propagation uses a raw similarity matrix, whereas Label Spreading minimizes a loss function that has regularization properties. All of the specifics for the implementation of these classes can be found in the scikit-learn documentation.

2.2.2 Wrapper

For the third semi-supervised learning method, I chose a simple wrapper method that I implemented myself. The basic synopsis of this method is that we first train the model on the training examples that we have the labels for, then we use this same model to generate labels for the unlabelled training data. With this new dataset containing both true labels and predicted labels from our first model, we train another model that will be used for evaluation on our validation dataset. I found that using K Nearest Neighbours for the initial label creation and Random Forest for the final training and validation testing gave the most interesting and best results.

3 Experimental Setup

Exploring and preprocessing the data that was provided is incredibly important to understand and gain additional information about the data as well as ensuring we do not introduce issues that may arise when training the models. Note that some of the preprocessing steps closely resemble those that were used in the paper provided[1] and are not of my own creation.

3.1 Missing Values

The handling of missing values is a very important part of preprocessing the data, below I have included table 3.1 which is similar to the table in the paper, where I show the features that are missing data as well as the percentage of data that is missing in order of decreasing percentages. Note that these percentages were taken from the original paper[1].

Feature	% Missing
Weight	97%
Medical Specialty	53%
Payer Code	52%
Race	2%
Diagnosis 3	1%

Table 1: Missing Values

3.1.1 Weight

First, the weight feature is clearly the feature with the most missing values. For the purposes of applying the previously discussed algorithms, it would not be wise to utilize this data as 98569 of the 101766 samples are marked as '?' which denotes a missing value, and thus the models would not be able to learn anything useful from this. However I do believe that there is something to be learned from this data. Below I have plotted the distributions of the target class for each weight grouping.

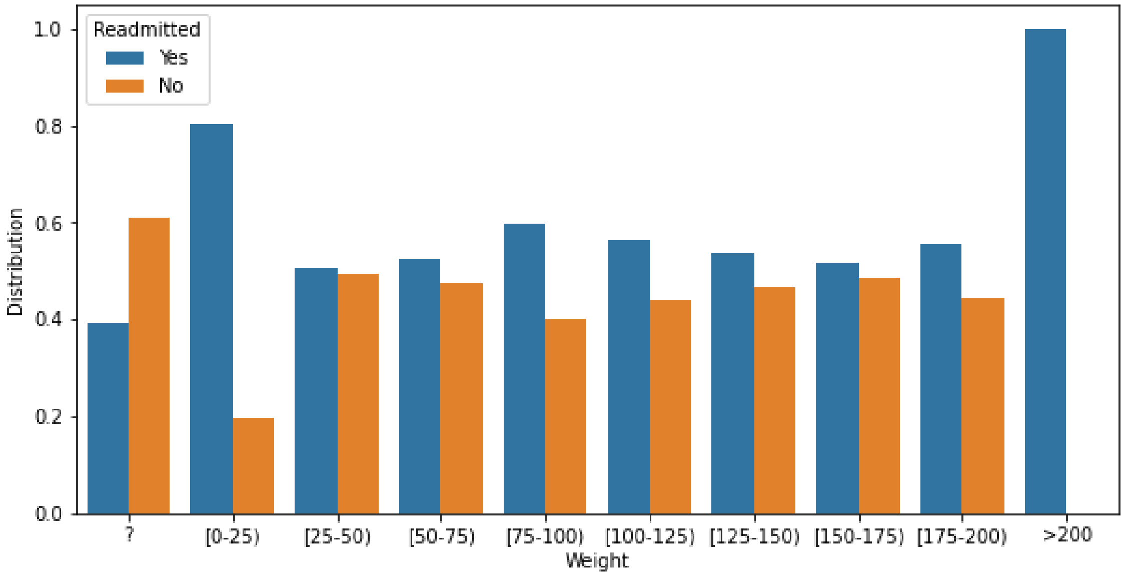


Figure 1: Distribution of readmission rate for weight groups

Something that I believe to be of note in this feature, is that both the '[0-25)' and '>200' weight groupings have a much larger distribution of patients that were readmitted into the hospital as opposed to not. A couple of things to note however before further analyzing this data. First, the distributions for these extremes are not comprised of very many samples and thus may not be representative of the real world. In addition to this, I also believe that for this weight value to be relatively useful, it needs to be accompanied by a height. While ideally, I would like to speak to a domain expert on the topic, from my very limited knowledge in the medical field and research it would seem that a higher body mass index (BMI) leads to more diabetes-related complications[2]. Overall, I believe that if there was significantly less missing data in this feature, it has potential to significantly improve predictions.

3.1.2 Medical Specialty

The next feature that contains 53% missing values is the medical specialty feature. While this feature is also missing a significant amount of data, I felt that there was enough data in this feature to keep it in the dataset that will be used to train the model. This was because regardless of this missing data, I felt that it would be beneficial to know which kind of doctor treated a patient, as this may give insights into what kinds of issues a patient who has diabetes may run into.

Once again, this is something that I would like to discuss with a domain expert, as I am not sure what leads to a particular type of medical specialist treating a person in a hospital. The goal of this would be to figure out if there is a definite link between this feature and different conditions that patients may have. In addition, while I do not know enough to say, there are quite a few medical specialties that seem like they may be able to be combined into one value. For example in the feature, there exists 'Oncology', 'Hematology/Oncology'and 'Hematology', I wonder whether or not these are equivalent or if there is a significant difference in the skill set of all three, for this reason I chose to not combine the three as my knowledge in the topic is very limited.

3.1.3 Payer Code

Similarly, payer code was missing 52% of the data within the feature. However, as opposed to the previous feature where I decided to keep this feature regardless of this missing data. In this case, I do not believe that payer code will affect the status of their condition and whether or not they will be readmitted. While there may be an argument to be made that in the united states, different payer codes may give information as to the income of the individual, which therefore may give an idea as to their quality of life outside of the hospital. This may have some kind of link to the readmission rates, however this would require more analysis and researching the link between payer codes and quality of life.

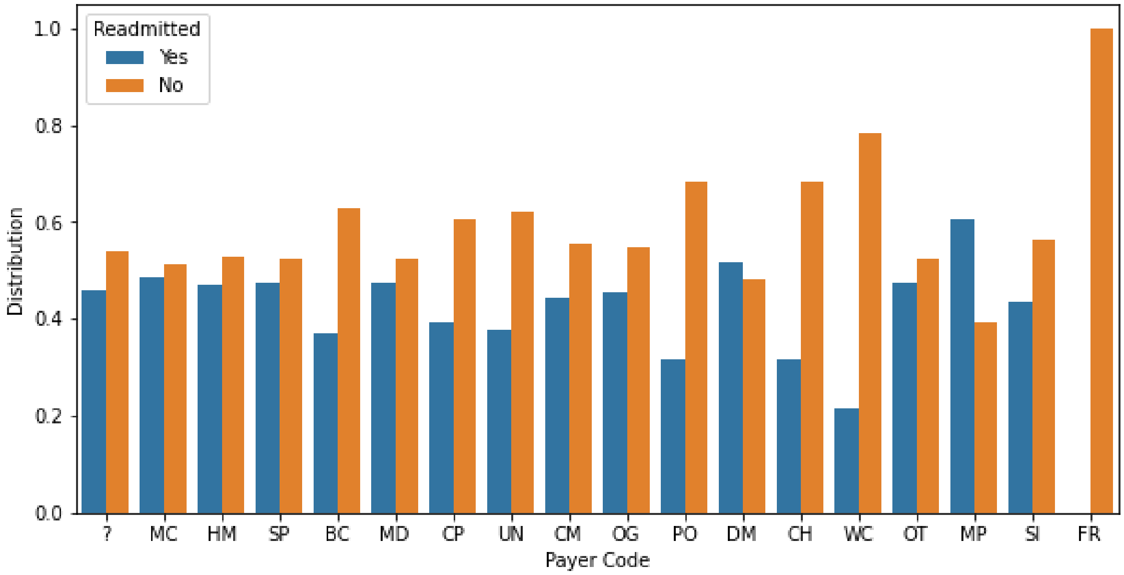


Figure 2: Distribution of readmission rate for payer code

This distribution is shown in figure 2, where we represent the ratios of the readmission feature on the different payer codes. As can be seen, most payer codes have a higher percentage of the 'No'value, however there are some exceptions, notably 'DM'and 'MP'. This could be due to implicit bias in the data, since these classes do not have a significant amount of data (neither over 600 values). We can also see the inverse, where payer codes such as 'BC', 'PO', and 'WC' have much higher rates of being released from the hospital, which is most likely due to external factors that may or may not be linked to this feature. For this reason, as I have said before, this data will not be used when training the models. However, there may be some truth to what is shown and I believe that we may be able to gain some information with further research into how payer codes work in different areas and whether or not there is a link between this feature and our target class.

3.1.4 Race & Diagnosis 3

Both of these features contained very small numbers of missing values (2% and 1% respectively). Because of this, both of these features were included in the data set, with the missing values included as a different encoded value (the specific encoding is referenced later in the paper.

3.2 Unneeded Features

If we now shift attention to the features which are not missing any values, there are some features that do not give us any useful information and may add unwanted bias.

3.3 Feature Encoding

The encoding of features and the technique used is very important for the techniques we will be using. The primary consideration when encoding features is the notion of distance, this is because for learners such as KNN, they function on this notion of distance. If we take a feature such as medical specialty, and reduce it down to 3 features: Type A, Type B and Type C. Performing an encoding technique such as label encoding on this dataset may lead to a mapping where {Type A, Type B, Type C} becomes {1, 2, 3}. While this may seem like it is acceptable at first, it introduces this notion of distance between values of this feature, where the distance Type A → Type C is a larger distance than Type B → Type C, which is not necessarily the case. The reasoning behind this is trying to avoid creating this false notion of distance between values of a feature. For this reason I choose one-hot encoding for some features where avoiding this notion of distance was deemed necessary. There are of course some exceptions, where I found that utilizing label encoding improved performance.

Feature	Encoding Technique
Gender	Label Encoder
Age	Label Encoder
Diagnosis 1	Label Encoder
Diagnosis 2	Label Encoder
Diagnosis 3	Label Encoder
Medications	Label Encoder
Glucose Serum	Label Encoder
A1C Test	Label Encoder
Medication Change	Label Encoder
Diabetes Medication	Label Encoder
Medical Specialty	One Hot Encoding

Table 2: Feature Encoding Techniques

As can be seen in table 3.3, most of the columns encoded use the label encoder, with the exception of medical specialty that uses one hot encoding. The main features that I experimented with one-hot encoding were the three diagnosis columns, where I found that using the label encoder method produced better results.

3.4 Duplicate Patients

In the main paper [1], they choose to remove duplicate patient visits. I chose to also do the same as I did not want to bias my results towards repeat patients. In my case, I simply used the first visit any of the patients had to the hospital. This allows me to assume that my model is not going to be biased towards repeat visitors but rather only consider the initial visit.

3.5 Sampling Method

Since the target class is slightly unbalanced after all of the preprocessing (more no revisit examples), I thought that using a sampling method to make it equal may increase performance. When choosing the sampling method, I did a lot of testing as to whether using the original dataset, undersampling, oversampling, or some kind of balanced sampling was best. In the end I chose oversampling because it lead to a much higher sensitivity. This works logically, as I am adding more examples in the positive class (readmitted is yes). For oversampling, I utilize the SMOTE technique that is implemented in scikit-learn.

4 Evaluation

4.1 Task 1 - Readmitted

4.1.1 Accuracy

The first step to figuring out which models are the best is looking at the accuracies in table 3. As can be seen from this table, our top two models are Random Forest and Naive Bayes, who outperform the other models by at least 6%. This however is not the full picture, as we can observe other metrics that give us more insights.

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbours	Naive Bayes	Decision Tree
1	0.685006	0.650808	0.608701	0.682331	0.615913
2	0.700942	0.641968	0.609282	0.687216	0.614284
3	0.694079	0.648831	0.604048	0.689078	0.614168
4	0.697453	0.627661	0.612539	0.696755	0.622310
5	0.695243	0.589043	0.616494	0.692683	0.608817
6	0.688612	0.629987	0.611841	0.690357	0.602303
7	0.691055	0.604630	0.617308	0.696406	0.613586
8	0.694894	0.645923	0.611725	0.691055	0.613935
9	0.693847	0.644643	0.613819	0.698151	0.613353
10	0.687449	0.632314	0.613935	0.688263	0.611958
avg	0.692858	0.631581	0.611969	0.691229	0.613063
std	0.004593	0.019250	0.003706	0.004657	0.004822

Table 3: Model Accuracy

4.1.2 Sensitivity & Specificity

The first two metrics that will give us more insights into how the models are working are sensitivity in table 4 and specificity in table 5. If we first look at sensitivity, we can see that our highest performing model is the KNN algorithm, however, this is where it becomes clear that there is this sensitivity-specificity tradeoff. We can see that despite KNN having the highest average sensitivity, it actually has the lowest specificity, meaning that it leans more on classifying the data as the positive class, rather than the negative class. Classifiers such as random forest and naive bayes tend to lean towards favouring the negative class. Overall, observing this tradeoff, I would still consider random forest and naive bayes to be the best performing models.

Something that I feel is important to comment on is the instability of the MLP classifier in relation to sensitivity and specificity. If we observe the standard deviation of this model, for both sensitivity and specificity it is significantly higher than other models. This is due to the MLP model heavily favouring either the positive or negative class, depending on the distribution of the training data in the fold. For example, fold 3 has a sensitivity of only 0.37, but has a specificity of 0.91, showing that for this fold, the mlp model favoured the negative class heavily. In contrast to this, fold 5 has a sensitivity of 0.88, but a specificity of only 0.28, suggesting that this model instead heavily favoured the positive class. It may be possible to increase the size of the hidden layer size, which may make it more consistent, however I will discuss why I did not do this more in the running time analysis.

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbours	Naive Bayes	Decision Tree
1	0.633099	0.462676	0.696948	0.625822	0.619484
2	0.639221	0.635233	0.698804	0.628665	0.629134
3	0.636385	0.371402	0.687117	0.633554	0.622463
4	0.650244	0.267255	0.695329	0.649314	0.628399
5	0.642562	0.880854	0.696051	0.638659	0.615014
6	0.634475	0.462329	0.700000	0.642466	0.611416
7	0.626936	0.811356	0.703895	0.635382	0.618724
8	0.636175	0.623932	0.689074	0.628552	0.608686
9	0.633918	0.562573	0.703158	0.646550	0.620351
10	0.630933	0.623524	0.700625	0.626765	0.616346
avg	0.636395	0.570113	0.697100	0.635573	0.619002
std	0.006147	0.178654	0.005244	0.008001	0.006261

Table 4: Model Sensitivity

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbours	Naive Bayes	Decision Tree
1	0.735993	0.835601	0.522020	0.737837	0.612405
2	0.761652	0.648593	0.521228	0.744808	0.599677
3	0.750172	0.918559	0.523285	0.743060	0.606102
4	0.744760	0.988822	0.529576	0.744294	0.616209
5	0.749352	0.289319	0.534780	0.748173	0.602452
6	0.744842	0.804126	0.520275	0.740100	0.592839
7	0.754095	0.401384	0.532180	0.756401	0.608535
8	0.754452	0.668229	0.533271	0.754452	0.619260
9	0.753124	0.725821	0.525451	0.749190	0.606432
10	0.744507	0.641187	0.526414	0.750351	0.607527
avg	0.749295	0.692164	0.526848	0.746867	0.607144
std	0.006768	0.206255	0.005030	0.005660	0.007367

Table 5: Model Specificity

4.1.3 F1 Score

The F1 Score gives us a tradeoff value between precision and recall, and is calculated based on equation 3. If we observe the F1 scores in table 6, it again says what we have observed in the previous results, where random forest and naive bayes outperform the other models.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbours	Naive Bayes	Decision Tree
1	0.665762	0.567684	0.638357	0.661292	0.615152
2	0.679466	0.637627	0.639476	0.665921	0.617972
3	0.672233	0.510459	0.631123	0.667661	0.613988
4	0.682689	0.418106	0.642405	0.681879	0.624841
5	0.681188	0.684751	0.647794	0.678040	0.614379
6	0.674924	0.560089	0.647587	0.678890	0.610370
7	0.668000	0.670480	0.645856	0.674807	0.613541
8	0.677407	0.639593	0.641230	0.672018	0.613576
9	0.673125	0.611570	0.644235	0.680537	0.614743
10	0.669780	0.630163	0.645822	0.668891	0.614781
avg	0.674457	0.593052	0.642388	0.672994	0.615334
std	0.005413	0.077094	0.004863	0.006577	0.003629

Table 6: Model F1 Score

4.1.4 Paired t-test

Finding whether there is a significant difference between our different models is an important part of analyzing our results. We can accomplish this by using what is known as the paired t-test. This test is computed via a few different steps. First, we compute the average and standard deviations of differences between two models, these values were taken from table 3. After these differences are computed, we can calculate what is known as the t-score which is defined by equation 4.

$$t = \frac{\bar{x}\sqrt{10}}{s} \tag{4}$$

Where $\bar{x}$ is defined as the average difference between the two models and s is defined as the standard deviation. Once we have the t-score, we can compute our p-value with 9 degrees of freedom using an online calculator which utilizes a lookup table (https://www.graphpad.com/quickcalcs/pvalue1.cfm). These corresponding values are shown in table 7.

Models	Avg	Std	t-score	p-value
RF-MLP	0.061277	0.021241	9.122869	< 0.0001
RF-KNN	0.080889	0.006409	39.911359	< 0.0001
RF-NB	0.001628	0.005443	0.946032	0.3688
RF-DT	0.079795	0.005730	44.041168	< 0.0001
MLP-KNN	0.019611	0.023406	2.649626	0.0265
MLP-NB	-0.059649	0.022838	-8.259235	< 0.0001
MLP-DT	0.018518	0.019549	2.995540	0.0151
KNN-NB	-0.079260	0.004101	-61.119277	< 0.0001
KNN-DT	-0.001093	0.006958	-0.496926	0.6312
NB-DT	0.078167	0.006585	37.535977	< 0.0001

Table 7: P-Value Calculations

After these values are calculated, we can determine whether two models are statistically significant by comparing them to some α value. In our case, we compare it to $\alpha = 0.05$, as we want a 95% confidence ($\alpha = 1 - 0.95$). From the p-values in table 7, we can see that the only models that are statistically different to each other according to our definition are K Nearest Neighbour and Decision Tree as well as Random Forest and Naive Bayes.

4.1.5 ROC Curves

We can also draw the ROC curves from each of the models. Drawing the ROC curve shows us the performance of each algorithm at each of the classification thresholds. The crucial part of these curves is the area under the curve (referred to as auc). This value is fairly straightforward to interpret; the higher the auc, the better the model performs. Note that with these curves, it only considers true and false positives, which is why it does not give us a complete picture of how the model performs.

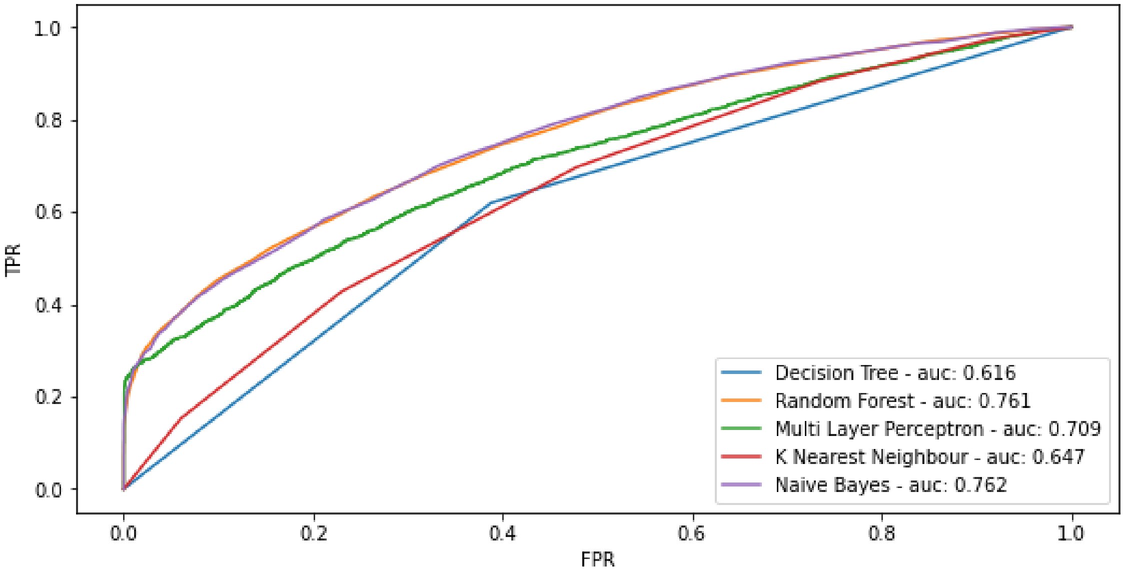


Figure 3: Model ROC Curves

Figure 3 shows the ROC curves of all of the models that we used on task 1. Note that because we have 10 different folds for each model, and therefore generating each models I do not show them all here, however I have included them in the appendix. In the case of figure 3, I include the first fold of each of the models, which closely resembles each fold of all models.

The first note from this graph is that our top two models are by far Random Forest and Naive Bayes, each of which achieve an auc of approximately 0.76. MLP is not far behind at 0.70, and KNN as well as Decision tree at 0.64 and 0.61 respectively, which is significantly worse than our other models.

Fold	Decision Tree	Random Forest	Multi Layer Perceptron	K Nearest Neighbour	Naive Bayes
1	0.616	0.7610	0.7090	0.6470	0.7620
2	0.614	0.7700	0.7100	0.6540	0.7630
3	0.614	0.7670	0.7080	0.6480	0.7670
4	0.622	0.7740	0.7180	0.6550	0.7760
5	0.609	0.7690	0.7060	0.6570	0.7690
6	0.602	0.7620	0.6990	0.6460	0.7650
7	0.614	0.7680	0.7030	0.6620	0.7680
8	0.614	0.7710	0.7110	0.6550	0.7700
9	0.613	0.7720	0.7070	0.6550	0.7700
10	0.612	0.7650	0.6960	0.6530	0.7650
avg	0.613	0.7679	0.7067	0.6532	0.7675

Table 8: AUC Values for Each Fold

If we observe table 8, this observation we made with graph 3 holds (where we only look at the first fold), where by far the best performing models are Random Forest and Naive Bayes. Therefore, we can determine that for this task, the best models are both Random Forest and Naive Bayes.

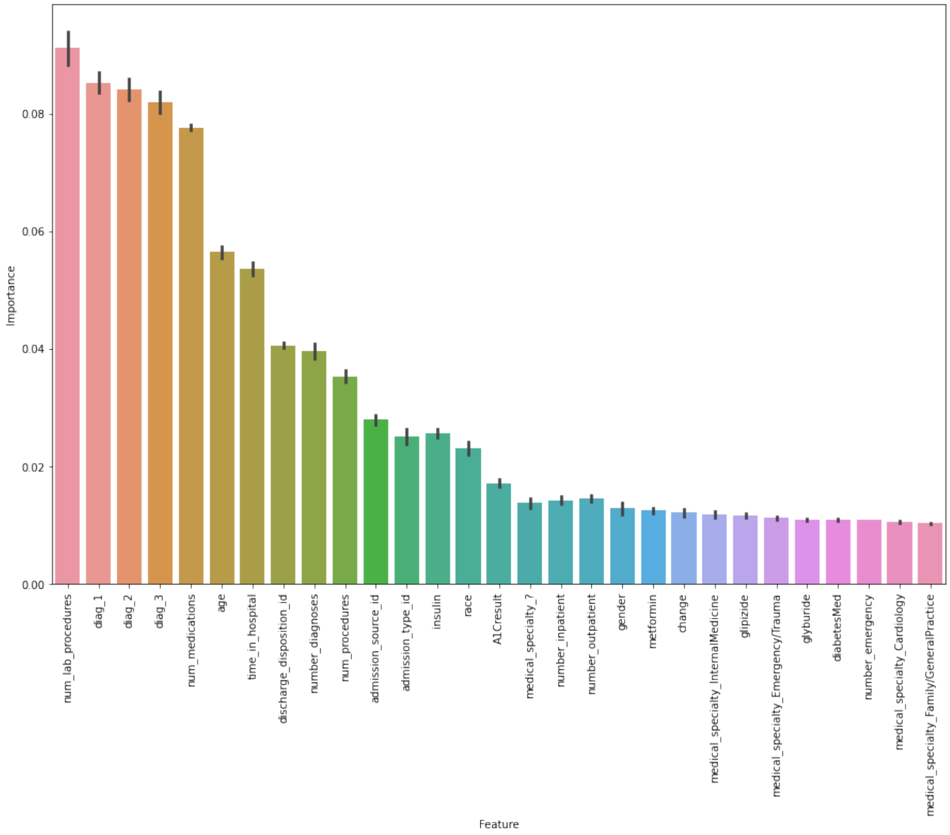


Figure 4: Feature Importances

4.1.6 Discussion

For this task, the clear two front runners for models are naive bayes and random forest. I would tend to prefer random forest as the way it is constructed it would be more robust to noisy features in the data. Therefore, I am comfortable saying that the best model for this task is random forest. Due to the performances that I was able to achieve predicting whether or not someone is admitted, it is not unreasonable to assume that there is a connection between these features and the target class. We can use the random forest feature importances in figure 4. Through this figure, it is shown that the most important features are the three diagnoses as well as the number of lab procedures and the number of medications. This is a common theme that will be seen throughout the report.

4.2 Task 2 - Gender

4.2.1 Accuracy

Again, for this task, we can first compare the accuracies of each of our models in table 9. Observing this table, we can see that Random Forest and Naive Bayes are our two top models, the same as in task 1. However something to note is that our accuracies are in general lower than the readmitted prediction. This tells me right away that there is not a clear difference between the two genders, as if there was, for example, a specific diagnosis that is much more common to one gender.

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbour	Naive Bayes	Decision Tree
1	0.612755	0.517291	0.561210	0.603156	0.563840
2	0.608810	0.562525	0.565943	0.616305	0.548060
3	0.619198	0.590007	0.563051	0.607758	0.546877
4	0.608679	0.548455	0.570414	0.610388	0.559895
5	0.616174	0.580539	0.565812	0.620644	0.555030
6	0.615122	0.565943	0.568310	0.612755	0.556476
7	0.617357	0.592373	0.566206	0.616174	0.549901
8	0.617883	0.546088	0.575016	0.620513	0.559632
9	0.614596	0.568442	0.566601	0.613281	0.559369
10	0.622091	0.572255	0.568442	0.613675	0.558317
avg	0.615266	0.564392	0.567101	0.613465	0.555740
std	0.004073	0.021428	0.003634	0.005153	0.005390

Table 9: Model Accuracies

4.2.2 Sensitivity & Specificity

Sensitivity and Specificity again are points of interest, due to the field. A difference from task 1 is that this time, KNN did not outperform the other models in sensitivity by a significant margin, but it still came with the same lower specificity. This further pushes the idea that it leans more on the positive class when classifying. Again the two models that outperformed the rest in general were Random Forest and Naive Bayes.

Once again, I feel it is important to note the results that I got from mlp, where they were incredibly inconsistent. In both specificity and sensitivity, there was a standard deviation of above 0.25, which is significantly higher than any of the other models. This again reinforces that phenomenon of mlp overfitting to one of the classes, while ignoring the other.

4.2.3 F1 Score

The F1 Score results in table 12 show the same as task 1, with the exception that they are significantly lower, further suggesting that the problem of gender prediction is much more difficult than that of readmission prediction.

4.2.4 Paired t-test

For the paired t-test, we again use the confidence rate of 95%, giving us an α value of 0.05. I again use equation 4 to calculate the t-value, then use an online calculator to determine the p-value. If the p-value is greater than α, the two models are statistically different, these calculations are shown in table 13.

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbour	Naive Bayes	Decision Tree
1	0.599137	0.951443	0.610197	0.587267	0.568654
2	0.585347	0.193297	0.599896	0.591842	0.549753
3	0.605532	0.361976	0.604726	0.592105	0.547798
4	0.583681	0.929093	0.609750	0.592544	0.564911
5	0.595112	0.444740	0.613974	0.591658	0.556323
6	0.584896	0.317236	0.603904	0.582584	0.567942
7	0.587218	0.616291	0.604243	0.592719	0.556574
8	0.597843	0.142013	0.609913	0.592450	0.562917
9	0.600643	0.615385	0.602519	0.594211	0.565532
10	0.595694	0.317931	0.614597	0.582568	0.565240
avg	0.593510	0.488941	0.607372	0.589995	0.560565
std	0.007282	0.269149	0.004734	0.004073	0.007104

Table 10: Model Sensitivity

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbour	Naive Bayes	Decision Tree
1	0.625705	0.104413	0.514623	0.618266	0.559261
2	0.632854	0.940895	0.531150	0.641374	0.546326
3	0.632311	0.808812	0.523061	0.622778	0.545993
4	0.634120	0.161051	0.530379	0.628549	0.554789
5	0.636813	0.713616	0.518615	0.649050	0.553762
6	0.646821	0.826778	0.530981	0.644397	0.544450
7	0.647742	0.568260	0.527858	0.639820	0.543174
8	0.638911	0.970089	0.538399	0.649960	0.556184
9	0.628033	0.523232	0.532008	0.631647	0.553433
10	0.648577	0.827450	0.522129	0.644889	0.551370
avg	0.637189	0.644459	0.526920	0.637073	0.550874
std	0.007776	0.289528	0.006804	0.010538	0.005235

Table 11: Model Specificity

As can be seen from table 13, the models that are statistically significant from one another are MLP and KNN, Random Forest and Naive Bayes, and MLP and Decision Trees. All of the other models are not statistically different from one another when trained on this dataset.

4.2.5 ROC Curves

The ROC curves in figure 5 tell us a similar story to that of our previous analysis where our two top performing models were random forest and naive bayes. Table 14 gives a more complete picture with all of the folds, where random forest is marginally higher, but not by any significant margin. As with task 1, I have included all of the roc curves in the appendix, however I feel that the first fold shown in figure 5 is a fair representation.

Fold	Random Forest	Multi Layer Perceptron	K Nearest Neighbour	Naive Bayes	Decision Tree
1	0.601327	0.657716	0.575499	0.590613	0.559671
2	0.602326	0.309034	0.583154	0.609580	0.551832
3	0.608966	0.463708	0.575444	0.596510	0.542121
4	0.600751	0.674872	0.588798	0.605407	0.564249
5	0.605487	0.512083	0.583291	0.606895	0.553090
6	0.608742	0.428002	0.588854	0.606336	0.567287
7	0.606438	0.602870	0.583091	0.607925	0.553890
8	0.615710	0.242650	0.595089	0.615200	0.566921
9	0.604614	0.583185	0.577002	0.601220	0.557390
10	0.612250	0.426784	0.587896	0.601681	0.561774
avg	0.606661	0.490090	0.583812	0.604137	0.557823
std	0.004589	0.136630	0.006171	0.006603	0.007438

Table 12: Model F1 Score

Models	Avg	Std	t-score	p-value
RF-MLP	0.050874	0.020905	7.695588	< 0.0001
RF-KNN	0.048166	0.005528	27.555204	< 0.0001
RF-NB	0.001801	0.006290	0.905627	0.3887
RF-DT	0.059527	0.007436	25.315447	< 0.0001
MLP-KNN	-0.002709	0.023185	-0.369452	0.7203
MLP-NB	-0.049073	0.020991	-7.392755	< 0.0001
MLP-DT	0.008652	0.027127	1.008631	0.3395
KNN-NB	-0.046364	0.004341	-33.775709	< 0.0001
KNN-DT	0.011361	0.005981	6.007086	0.0002
NB-DT	0.057725	0.008688	21.011440	< 0.0001

Table 13: P-Value Calculations

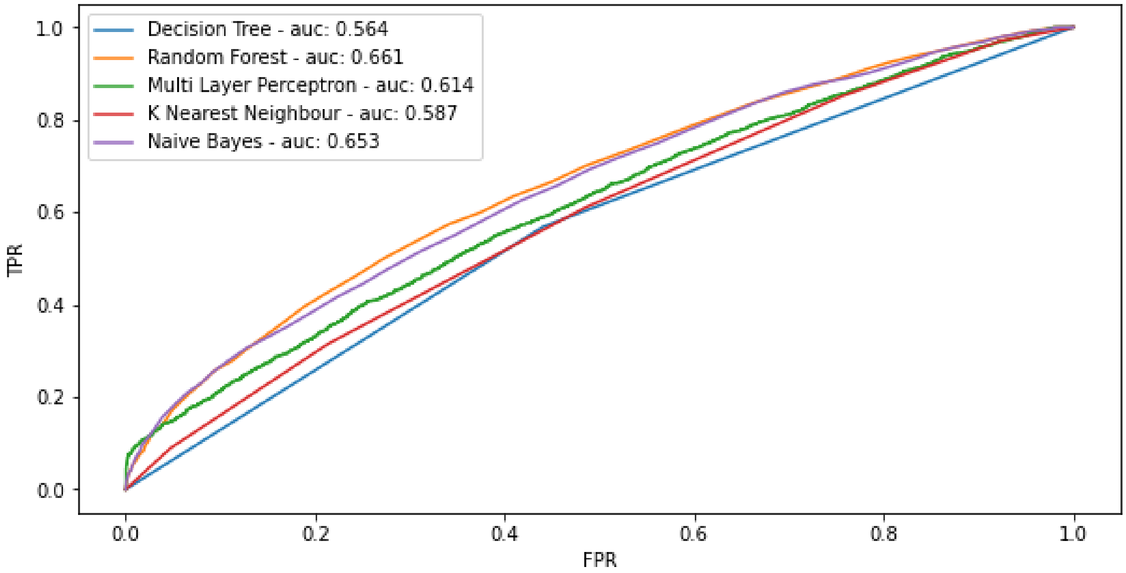


Figure 5: Model ROC Curves

Fold	Decision Tree	Random Forest	Multi Layer Perceptron	K Nearest Neighbour	Naive Bayes
1	0.5640	0.6610	0.614	0.5870	0.6530
2	0.5480	0.6620	0.635	0.5910	0.6650
3	0.5470	0.6670	0.639	0.5940	0.6660
4	0.5600	0.6620	0.634	0.6010	0.6640
5	0.5550	0.6660	0.622	0.5880	0.6690
6	0.5560	0.6640	0.616	0.5880	0.6660
7	0.5500	0.6660	0.638	0.5930	0.6630
8	0.5600	0.6700	0.608	0.5940	0.6740
9	0.5590	0.6660	0.612	0.5910	0.6650
10	0.5580	0.6730	0.622	0.5960	0.6700
avg	0.5557	0.6657	0.624	0.5923	0.6655

Table 14: AUC Values

4.2.6 Discussion

The results of running this experiment is similar to that of task 1, where random forest appears to be the best option for predicting the gender of a patient based on their treatment data. We can also deduce based on the reduced performance, that there is not a significant difference in what issues different genders face when admitted to the hospital. However, there is at least a small difference, as the models do perform better than random guessing.

Figure 6 shows the feature importances for each model (we only show features that have an importance higher than 0.01). It can also be noted that none of the features have a higher importance than 0.1 (10%), showing that there is not specifically one feature that can differ either gender. However, what this does suggest is that these small differences lie in the three diagnoses that are made, as well as the number of lab procedures. I think it would be interesting to talk to a subject matter expert and determine if they look for particular diagnoses, or administer certain lab procedures in particular genders.

4.3 Supervised Model Running Time

The running times for each of the algorithms tended to be rather fast. Note that before I analyze these results, I was running all of my tests on google colab pro, and while it is typically very stable, the CPU's that are provided are typically not incredibly fast. Table 15 shows the running times for each of our algorithms, there is not a significant difference between any of the models. The only notable difference is that MLP typically takes longer to train than the other models.

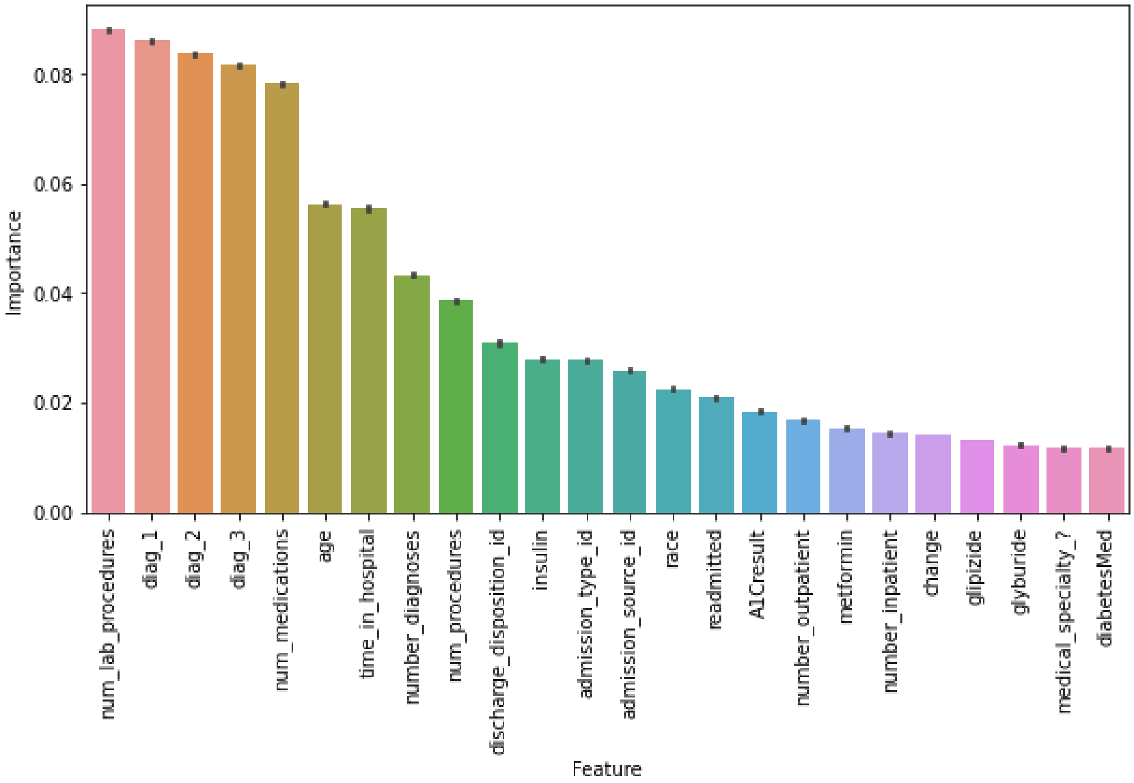


Figure 6: Feature Importances

Model	Task 1 Runtime	Task 2 Runtime
Decision Tree	00:21	00:26
Random Forest	03:22	03:34
MLP	10:59	07:27
KNN	04:06	01:24
Naive Bayes	03:42	03:28

Table 15: Supervised Model Runtime

Previously in the report, I mentioned that I did not use a larger hidden layer size. This was primarily due to the running time of such a model. I attempted to increase the hidden size to two, 100 neuron size hidden layers. This however lead to a massive increase in the run time, taking approximately 1hr30min to run. Overall, I did not consider the marginal increase in performance worth this massive increase in run time, however later in the report I will discuss other implementations that may be possible.

4.4 Semi-Supervised Learning

As stated earlier in the report, our semi-supervised learning task is split into three models; Label Propagation, Label Spreading, and a simple Wrapper. We experiment on various missing value percentages of 0% (a simple baseline), 10%, 20%, 50%, 90%, and 95%.

4.4.1 Accuracy

For each model, we have the average accuracies over all of the folds, see the appendix for a full list of values over all folds. We can observe for all three models, as we move from low to high ratios of missing labels, the accuracy generally decreases. Figure 7 shows this in a graph representation, where it becomes very clear that the wrapper implementation consistently has a higher accuracy at every missing value ratio.

Missing Value Ratio	Label Propagation	Label Spreading	Wrapper
0	0.601186	0.600547	0.692858
0.1	0.596289	0.596010	0.687077
0.2	0.591834	0.591392	0.681901
0.5	0.556613	0.574654	0.649564
0.9	0.501082	0.532907	0.554182
0.95	0.500209	0.524590	0.540060

Table 16: Accuracies Per Missing Value Ratio

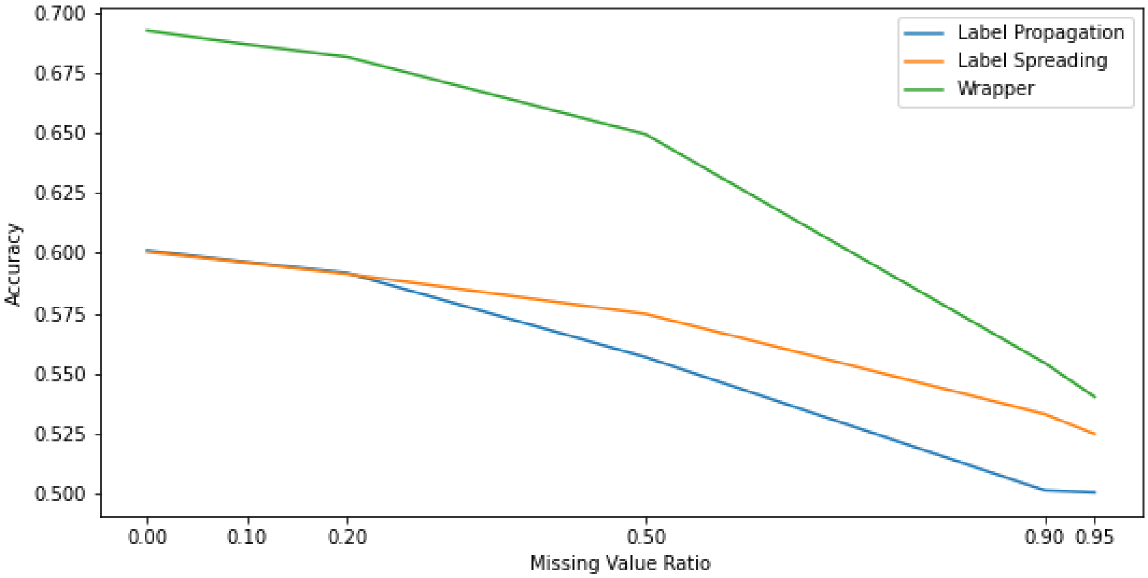


Figure 7: Accuracies v. Missing Value Ratio

4.4.2 Sensitivity & Specificity

Table 17 and 18 show the sensitivities and specificities over all models and missing value ratios. One of the key results from this is how Label Propagation acts at very high missing values. If we refer back to table 16, we can see that at 90% and 95% missing labels, the accuracy is almost 50%, which is rather close to random guessing. If we now observe the sensitivity in table 17, we can see that at those percentages, the sensitivity drops to almost 0%. In contrast to this, the specificity rises to nearly 100%. This suggests that at these higher missing label rates, the classifier is classifying almost all examples as belonging to the negative class.

Label Spreading and Wrapper have less interesting results. They both tend to lean more towards the positive class, with both having specificity dropping below 50%. There are a couple of abnormal points however that act strange. If we look at the wrapper sensitivity, it increases from 0% to 50%, before dropping again when we try above 90%. This is most likely due to us reaching the limit on the KNN classifier, and incorrectly classifying examples, leading to the random forest classifier incorrectly classifying.

Missing Value Ratio	Label Propagation	Label Spreading	Wrapper
0	0.684966	0.686570	0.636395
0.1	0.672359	0.687871	0.657801
0.2	0.670633	0.682207	0.675292
0.5	0.438464	0.668890	0.730790
0.9	0.004052	0.593861	0.679848
0.95	0.000931	0.552876	0.635490

Table 17: Sensitivity Per Missing Value Ratio

Missing Value Ratio	Label Propagation	Label Spreading	Wrapper
0	0.517402	0.514519	0.749295
0.1	0.520211	0.504152	0.716341
0.2	0.513036	0.500571	0.688469
0.5	0.674744	0.480354	0.568266
0.9	0.998113	0.471962	0.428585
0.95	0.999489	0.496258	0.444584

Table 18: Specificity Per Missing Value Ratio

4.4.3 F1 Score

Table 19 shows the F1 scores for each of the models, again calculated using equation 3. This table reinforces what we saw with the sensitivity and specificity tables. What is interesting as well, is that with the wrapper model, the F1 score increases up until 0.5 where it decreases again. Our observations about the label propagation model are also reinforced, where the F1 score drops to almost 0 when we have more than 90% unlabelled data.

Missing Value Ratio	Label Propagation	Label Spreading	Wrapper
0	0.631997	0.632166	0.606661
0.1	0.624804	0.629981	0.677616
0.2	0.621633	0.625393	0.679758
0.5	0.497116	0.611254	0.675875
0.9	0.008056	0.559718	0.603904
0.95	0.001859	0.537661	0.580080

Table 19: F1 Score Per Missing Value Ratio

4.4.4 Paired t-test

As we have done previously in the report, we can use the paired t-test in order to determine whether or not there is a statistical difference in the model results. Taking the differences in all of our model accuracies and calculating the average and standard deviation, table 20 is generated using this data.

Models	Metric	0	0.1	0.2	0.5	0.9	0.95
L.P. - L.S.	avg	0.000640	0.000279	0.000442	-0.018041	-0.031825	-0.024381
	std	0.001008	0.002150	0.001253	0.004089	0.007621	0.007734
L.P. - Wrapper	avg	-0.091672	-0.090787	-0.090066	-0.092951	-0.053100	-0.039851
	std	0.009611	0.008010	0.006291	0.003736	0.008414	0.009025
L.S. - Wrapper	avg	-0.092311	-0.091067	-0.090508	-0.074910	-0.021275	-0.015471
	std	0.009324	0.007581	0.006032	0.005397	0.005941	0.007320

Table 20: Accuracy Differences

Using this data in table 20, we can use equation 4 once again to calculate the t scores in table 21. Once we generate these t scores, we can use them with an online calculator in order to generate the P values in table 22. We once again use 9 degrees of freedom (10 folds - 1), and with a confidence of 95%. Therefore we compare our p values against an α of 0.05 and we can determine that label propagation and label spreading differ when there is 0%, 10%, and 20% missing data. There are not any other combinations of models that are statistically different.

Models	0	0.1	0.2	0.5	0.9	0.95
L.P. - L.S.	2.006088	0.410632	1.115932	-13.952855	-13.205361	-9.968102
L.P. - Wrapper	-30.162412	-35.840102	-45.271070	-78.682935	-19.957006	-13.964055
L.S. - Wrapper	-31.308771	-37.988694	-47.446592	-43.890958	-11.324866	-6.683501

Table 21: T Scores

Models	0	0.1	0.2	0.5	0.9	0.95
L.P. - L.S.	0.0758	0.6909	0.2934	< 0.0001	< 0.0001	< 0.0001
L.P. - Wrapper	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
L.S. - Wrapper	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001

Table 22: P Values

4.4.5 ROC Curves

I decided that it was more appropriate to draw separate ROC curves for the different mdoels in this case, since we are dealing with multiple ratios of missing values. One note I will make is that in the case of figure 8, where we show the label propagation curve, it seems as if the scenario where there are 95% labels missing, the auc is higher than the other scenarios. To show why this is not the case, I have also included figure 11, which shows the instability of this model at high ratios of missing labels, where the auc ranges from 0.8 to 0.25.

Our other two models shown in figure 9 and figure 10 show what we expect, where as we remove labels, the auc reduces and the roc curve moves more towards the diagonal. Looking at purely these roc curves, the clear option for best model is the wrapper, where the auc reduces to only 0.56 at 95% unlabelled data and 0.76 at the baseline. This is in contrast to label spreading which only has a baseline value of 0.64 and reduces to 0.53 at 95% unlabelled data.

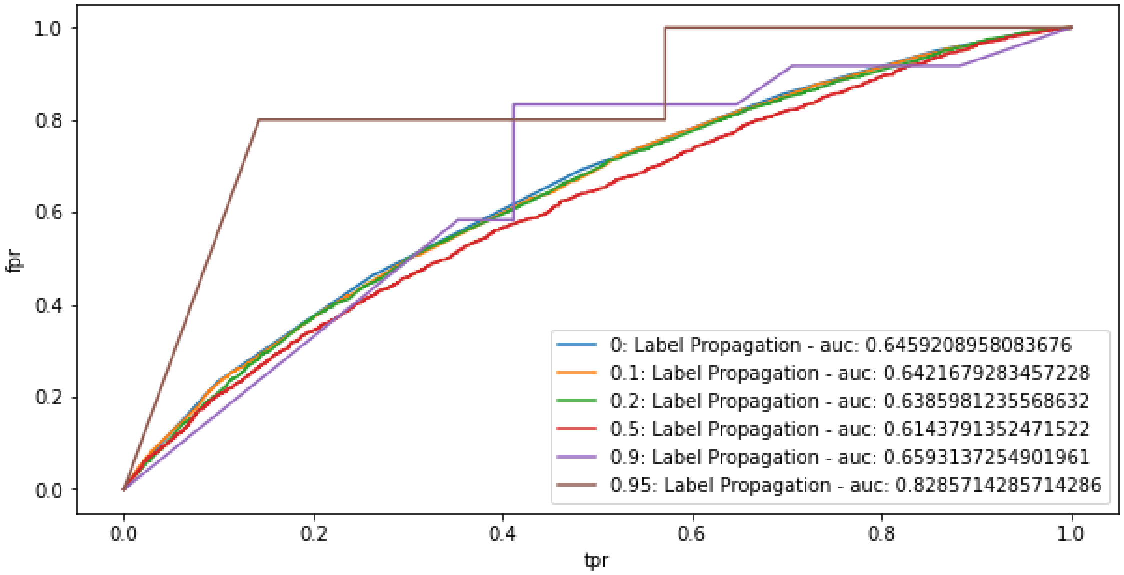


Figure 8: Label Propagation ROC Curves

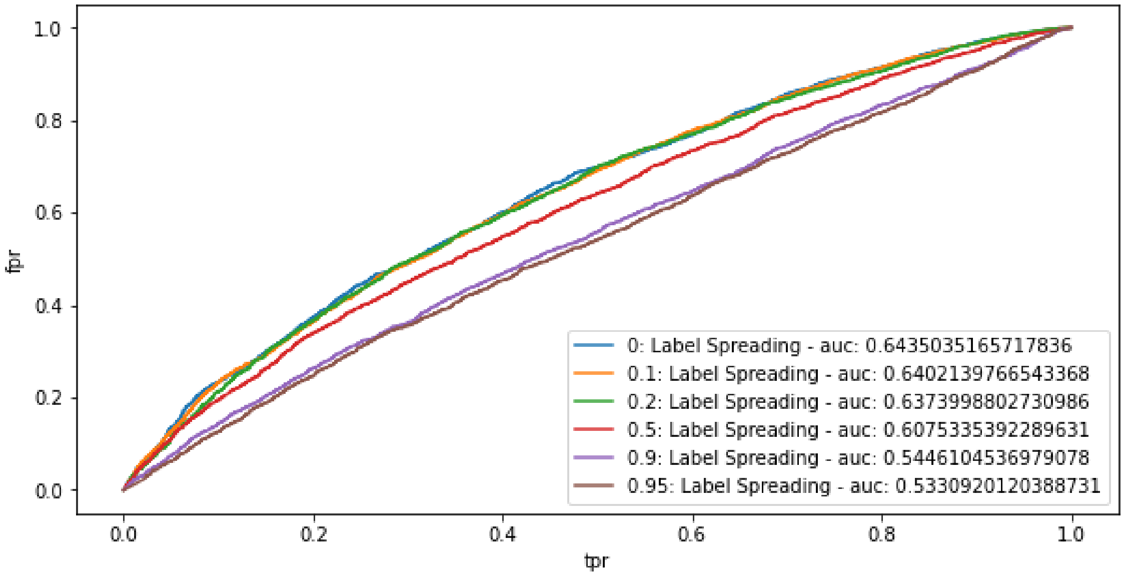


Figure 9: Label Spreading ROC Curves

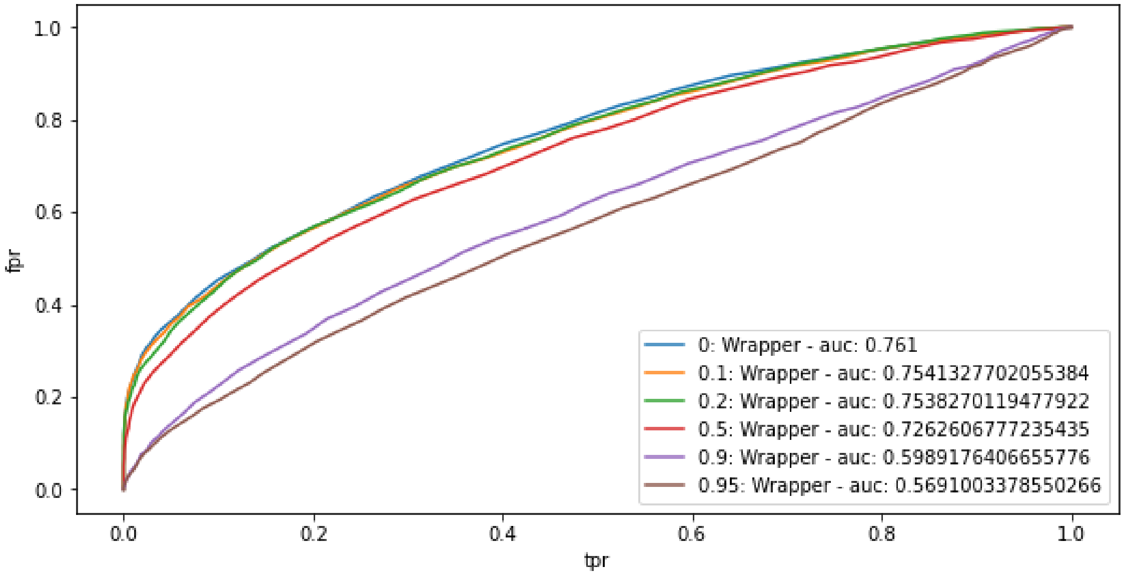


Figure 10: Wrapper ROC Curves

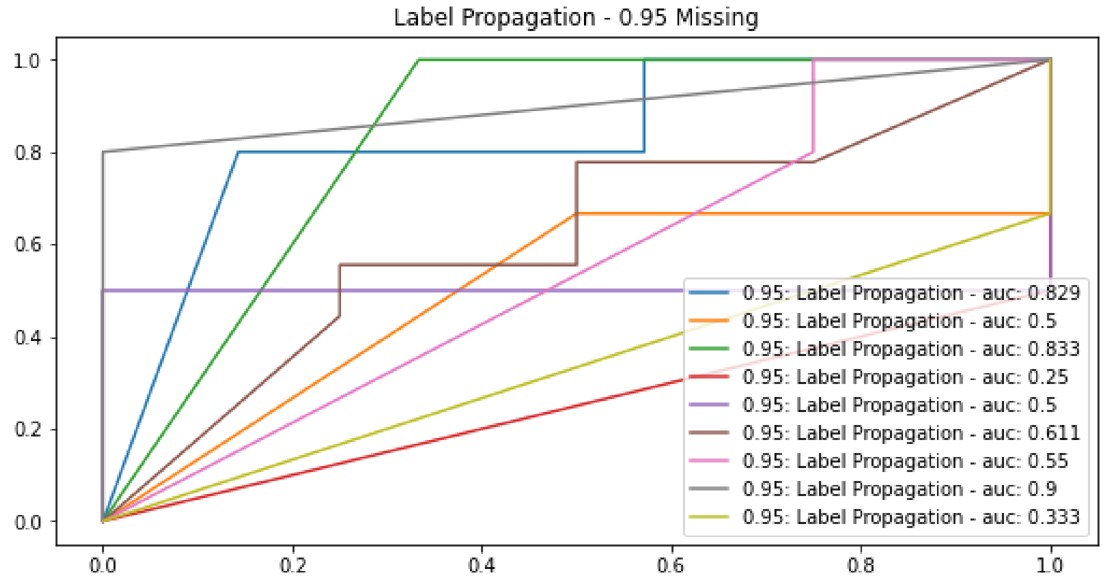


Figure 11: Label Propagation ROC Curve - 95% Missing Labels

4.4.6 Running Time

The running times of the algorithms was much less interesting this time as compared to the supervised learning portion of this report. In this case, all algorithms varied between 30-35 minutes of runtime depending on how fast the cloud server was at that particular time. This was the time to run the tests for all missing value ratios, and compute accuracies, recalls, and f1 scores. Since these computations take some time, it's not unreasonable to say that the training may have only taken approximately 25 minutes, but this difference is consistent among all models. In addition, for the wrapper this time includes training the KNN classifier.

4.4.7 Discussion

I think there are several takeaways from analyzing this data. The first being that, by far the best performing model was the simple wrapper. This was shown in all of our evaluation metrics. In addition to this, Label Propagation was by far the worst performing model. This model fell apart at above 90% missing labels. While it did not perform as well as I thought it would, I think that some of this performance can potentially be fixed. When I made preprocessing choices, I was attempting to balance performance, while trying to prevent running times becoming very large with extremely high dimensional data. If I were to no longer consider running time in my analysis, it would be possible to use one-hot encoding for, at minimum, the three diagnosis features. This would increase the number of features in our dataset to a massive amount, however this similarity matrix that is generated would be more representative of the actual data. As I stated previously in the report, label encoding can add this extra notion of distances, which can throw off many distance-based classifiers.

5 Synthesis

5.1 Lessons Learned

I learned many lessons working with this data. The first, and perhaps most important lesson that I learned was that sometimes a simple model is better than a more complex model. As I stated in my supervised learning evaluations, naive bayes performed almost as well as random forest. I did not at all anticipate naive bayes being able to handle the large amount of features that were present, and make very accurate predictions. This tells me that these traditional methods can still perform well when it comes to complex data, competing with more complex and modern approaches such as random forest.

Another lesson that was learned was the incredible importance of different evaluation methods. There were times when I was tuning preprocessing methods, it was important to check all of accuracy, sensitivity, and specificity, as it gives different perspectives on how a model evaluates data, and whether a particular model tends to overclassify the positive or negative class.

5.2 MLP

In this report, I mentioned the difficulties with the MLP model multiple times. I fully believe that there are configurations of fully connected neural networks that will lead to better performance, however I also believe that it is out of scope for this report, and could potentially be it's own report by itself. This would first, involve using tensorflow or pytorch which would allow for GPU processing to be used, this would solve the issue that I ran into with this report of very long runtimes. I have a high level of confidence that with these configurations in the new implementations, we could increase our initial MLP scores, as well as potentially overtake the random forest results that we observed.

However, this also comes with another problem, as it does not give us any level of being able to understand how the network comes to these solutions, it acts as a 'black box'. This is a problem that man neural networks face, where it is incredibly difficult to determine how the network is interpreting the input data, particularly in large networks. This explainability is rather important in the health care domain, as a doctor would most likely want to know why a model makes a particular decision rather than trusting it implicitly, since the consequences of getting something wrong could lead to a patient death.

As such, I do believe that MLP is an interesting point of focus, and could potentially achieve scores that are very good. However, for a real world solution that doctors may actually use, in my opinion it is not suitable for these kinds of scenarios, but could be used in tandem as a 'warning', alerting doctors that someone may be at risk, but not as something they should be very confident in.

5.3 Random Forest

I believe that Random Forest has the highest potential in this use case. This model has some advantages that I think are important to a model that is going to be used in this industry. Out of the models that I tested, it gave us the best results. I am confident that if more combinations of hyperparameters were tried, higher scores could be obtained.

6 Conclusion

Overall, I think there are a few conclusions that can be made from the analysis. The first, and perhaps most important is that, through the analysis of task 1, the most important features are the diagnoses that the patient receives, as well as the number of lab procedures and the number of medications. Through these features, we were able to effectively predict whether or not someone would be readmitted, with approximately 70% accuracy.

Task 2 allowed us to explore whether we could find a difference in the reasons that genders are admitted and treated in hospital such that we could predict what gender someone is based solely on their hospital visit details. Through this, we were able to achieve approximately 60% accuracy, which is not high enough to suggest significant differences, however what it does tell us is that there are subtle differences.

The semi-supervised learning problem demonstrated that despite missing labels from 50% of the data, we can still achieve 64% accuracy. This is a significant accuracy for how many labels are missing.

As I have said in this report, I believe that both the performance and understanding of these algorithms can be helped by an expert in the field. I would particularly be interested in how they approach treating different genders, if different genders typically display different symptoms, and if certain diagnoses are more at risk of readmittance than others, such as something that is chronic vs acute. I believe that an expert in this field may be able to provide insights into this, or point me to other datasets that could allow for further analysis.

References

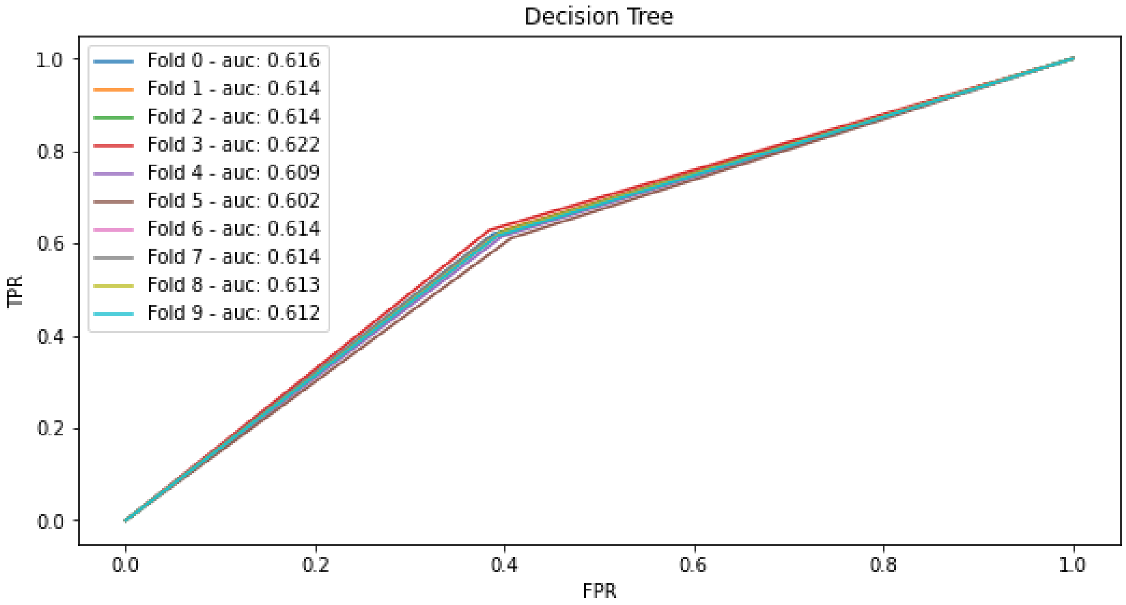
[1] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore, "Impact of hba1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records," *BioMed Research International*, vol. 2014, p. 781670, Apr 2014.

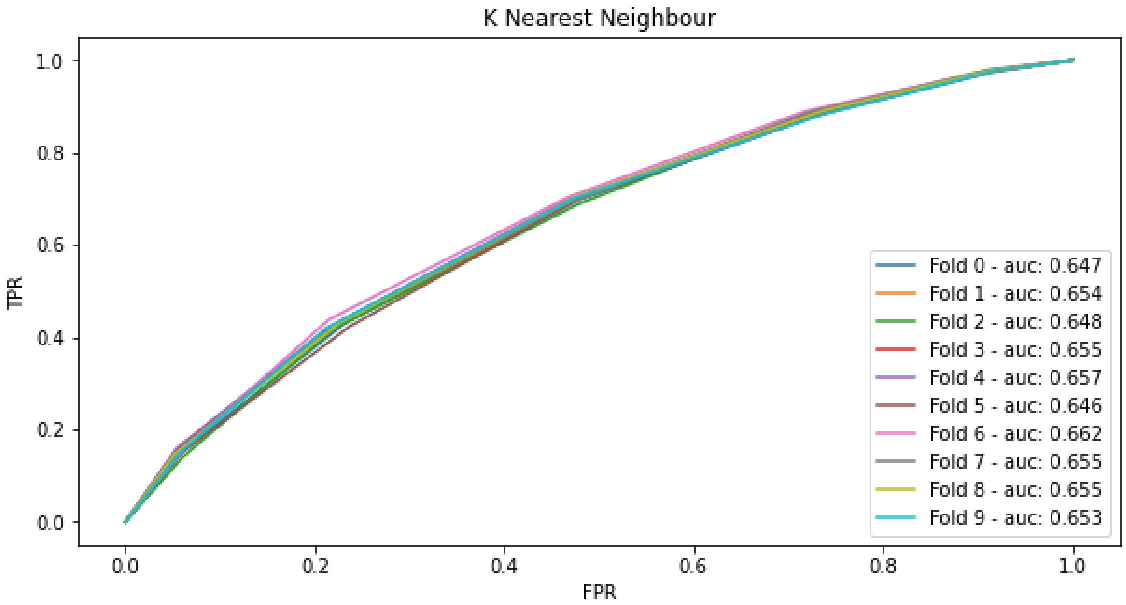
[2] A.-S. Alexopoulos, M. Fayfman, L. Zhao, J. Weaver, L. Buehler, D. Smiley, F. J. Pasquel, P. Vellanki, J. S. Haw, and G. E. Umpierrez, "Impact of obesity on hospital complications and mortality in hospitalized patients with hyperglycemia and diabetes," *BMJ open diabetes research & care*, vol. 4, pp. e000200–e000200, Jul 2016.

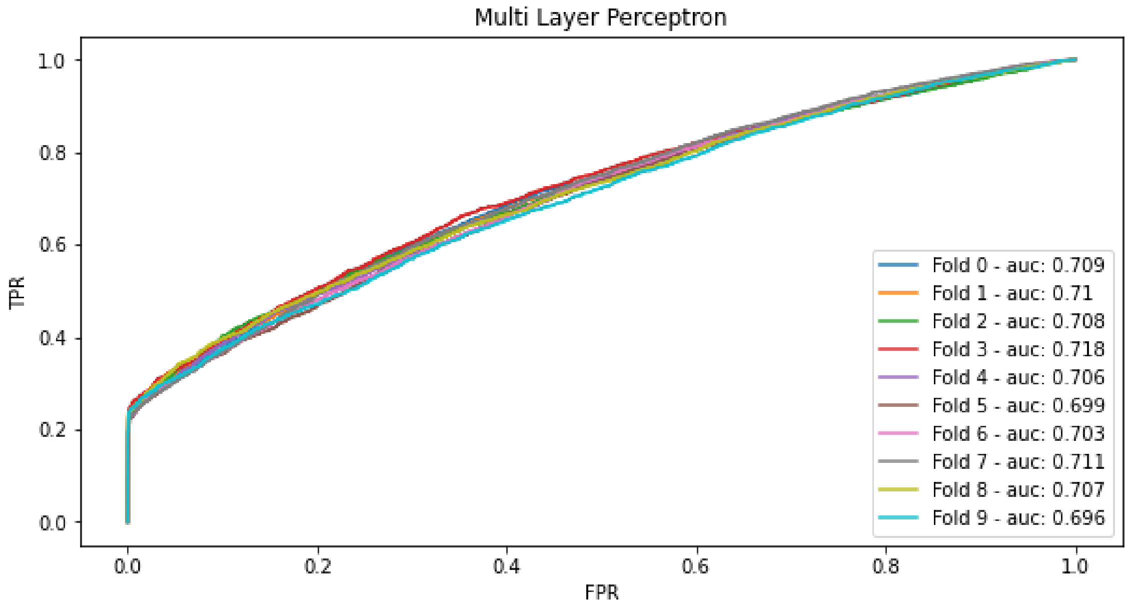
7 Appendix

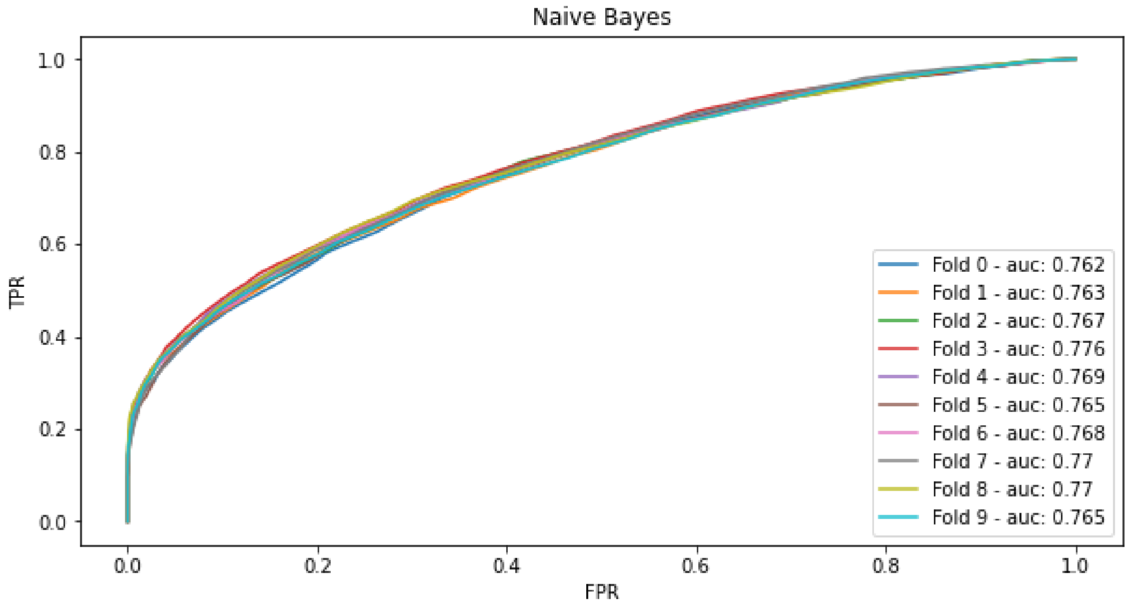
7.1 Supervised Learning

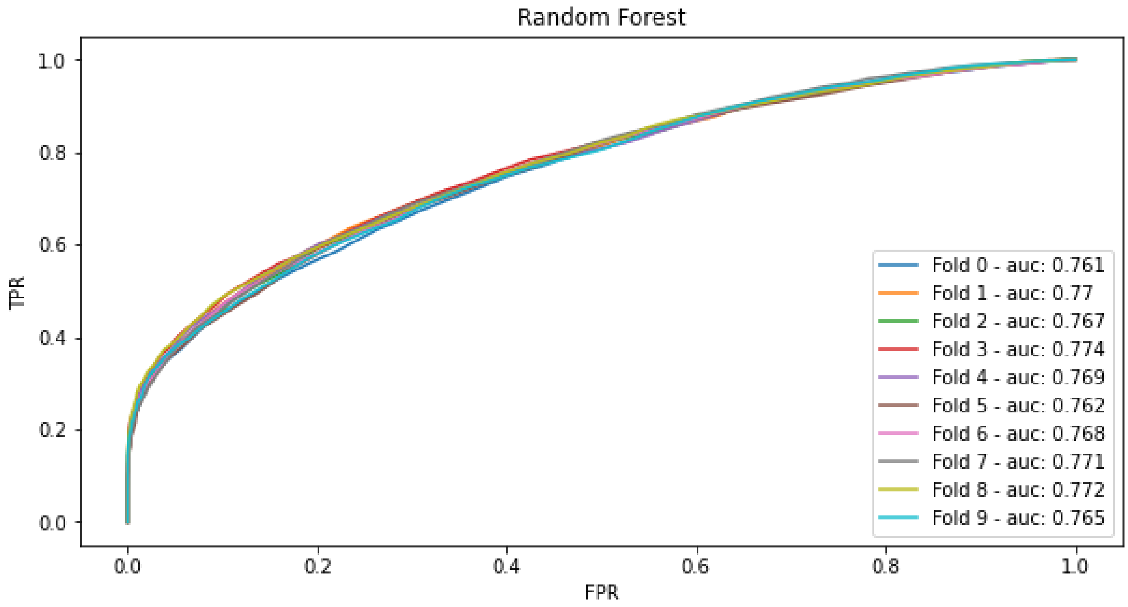
7.1.1 Task 1 - ROC Curves



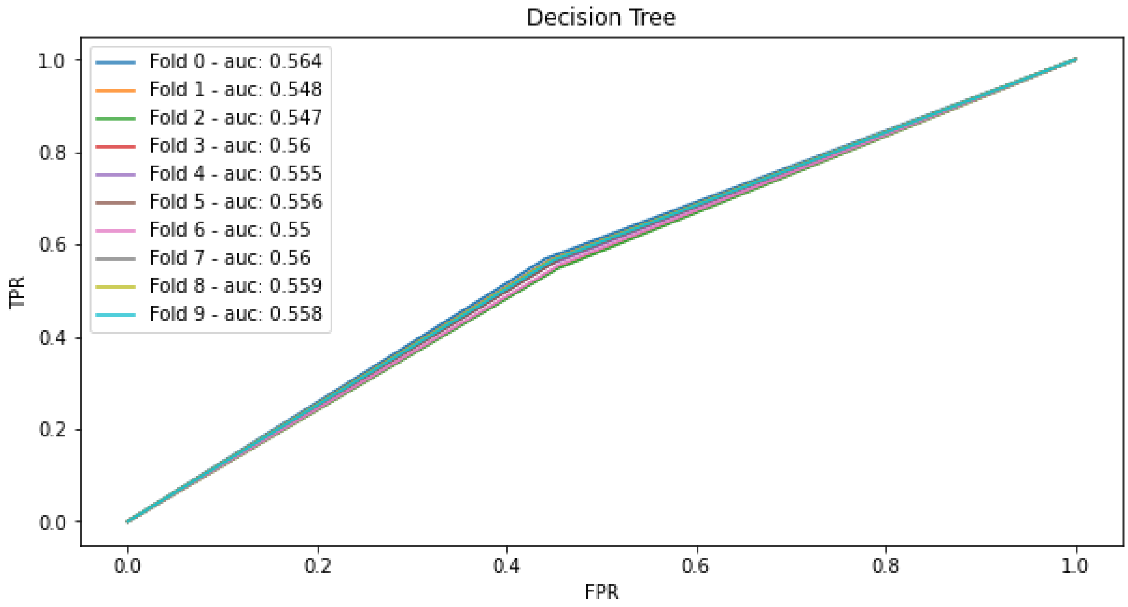


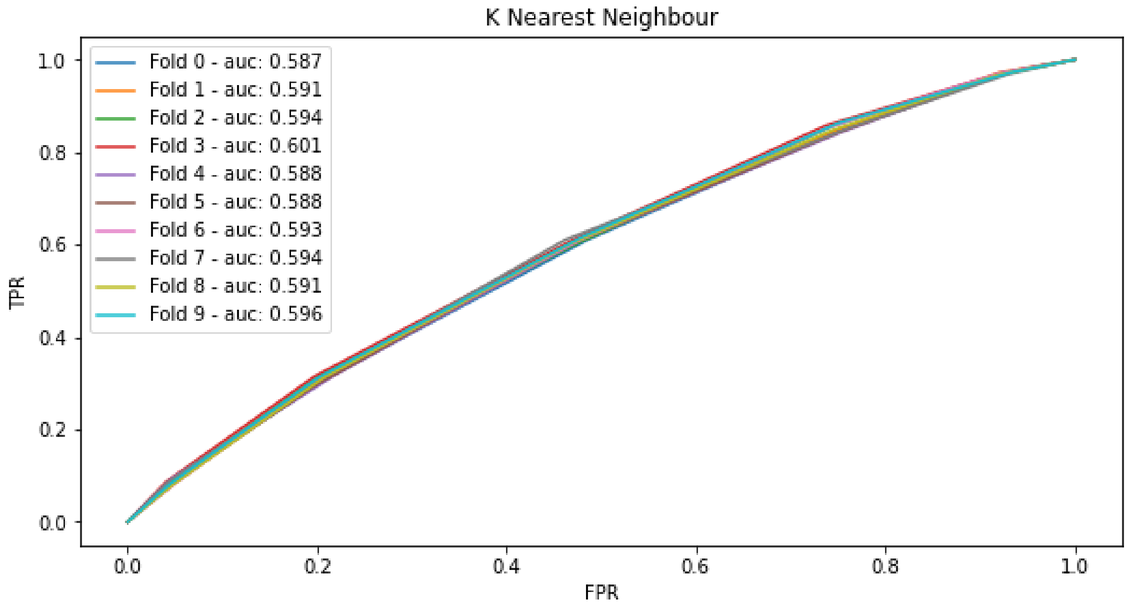

Multi Layer Perceptron
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Fold 0 - auc: 0.709
Fold 1 - auc: 0.71
Fold 2 - auc: 0.708
Fold 3 - auc: 0.718
Fold 4 - auc: 0.706
Fold 5 - auc: 0.699
Fold 6 - auc: 0.703
Fold 7 - auc: 0.711
Fold 8 - auc: 0.707
Fold 9 - auc: 0.696

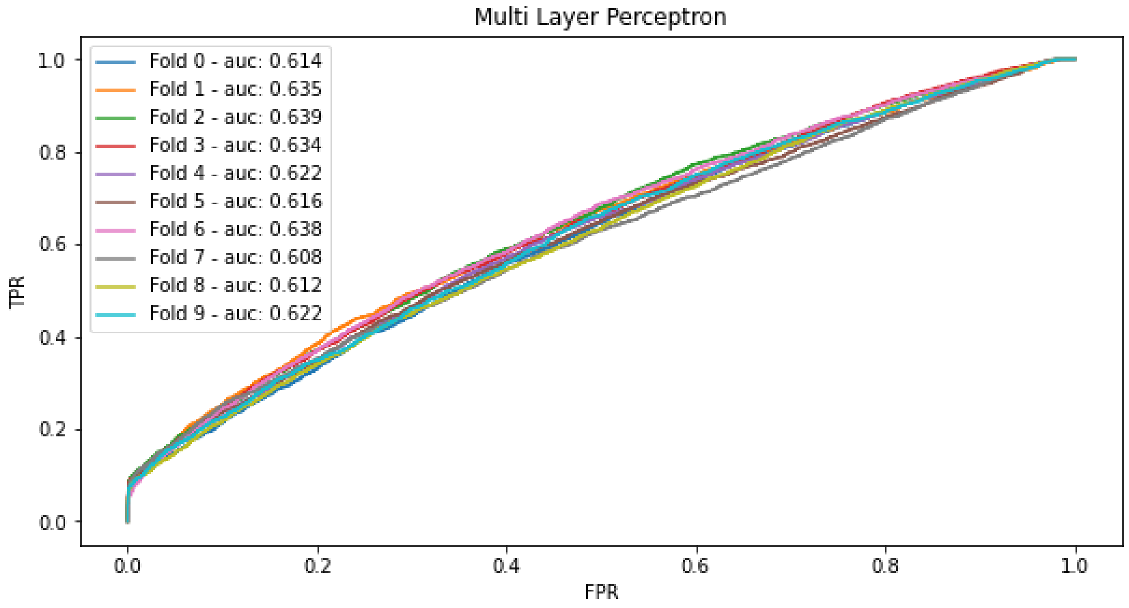

Naive Bayes
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Fold 0 - auc: 0.762
Fold 1 - auc: 0.763
Fold 2 - auc: 0.767
Fold 3 - auc: 0.776
Fold 4 - auc: 0.769
Fold 5 - auc: 0.765
Fold 6 - auc: 0.768
Fold 7 - auc: 0.77
Fold 8 - auc: 0.77
Fold 9 - auc: 0.765

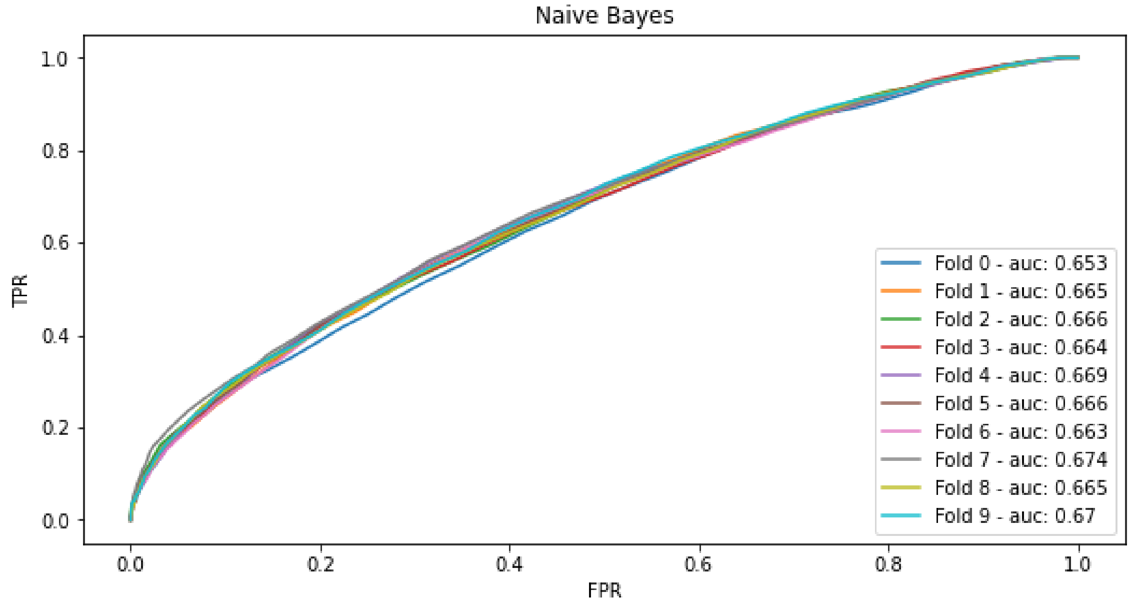


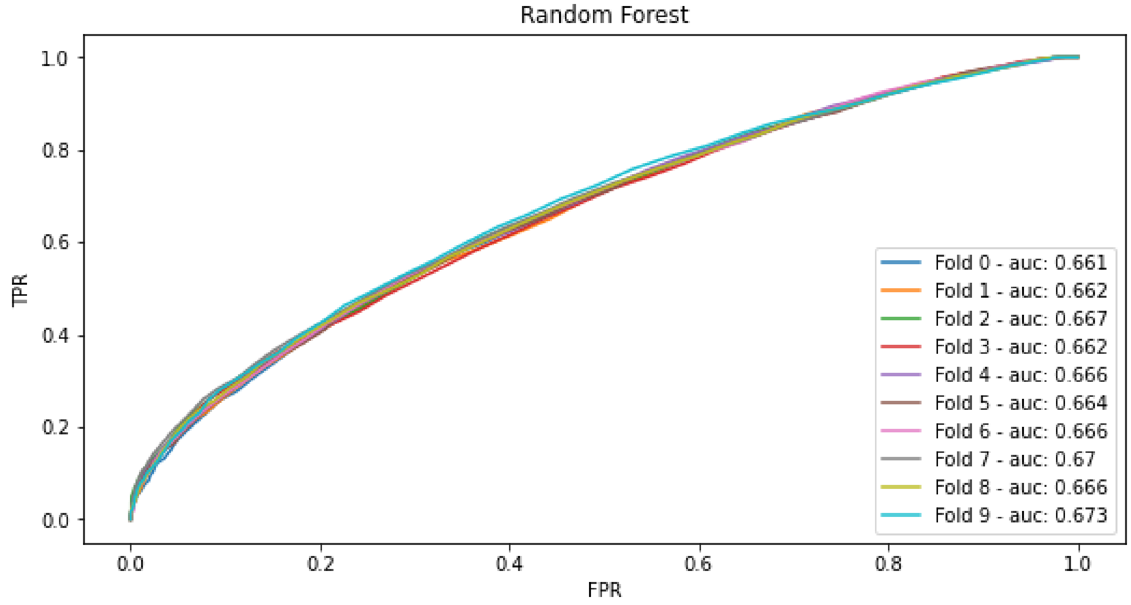
7.1.2 Task 2 - ROC Curves




K Nearest Neighbour
Fold 0 - auc: 0.587
Fold 1 - auc: 0.591
Fold 2 - auc: 0.594
Fold 3 - auc: 0.601
Fold 4 - auc: 0.588
Fold 5 - auc: 0.588
Fold 6 - auc: 0.593
Fold 7 - auc: 0.594
Fold 8 - auc: 0.591
Fold 9 - auc: 0.596
TPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
FPR


Multi Layer Perceptron
Fold 0 - auc: 0.614
Fold 1 - auc: 0.635
Fold 2 - auc: 0.639
Fold 3 - auc: 0.634
Fold 4 - auc: 0.622
Fold 5 - auc: 0.616
Fold 6 - auc: 0.638
Fold 7 - auc: 0.608
Fold 8 - auc: 0.612
Fold 9 - auc: 0.622
TPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
FPR





7.2 Semi-Supervised Learning

For the wrapper technique with 0 missing values, refer to task 1 random forest values.

7.2.1 Full Accuracy Tables

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.605327	0.596720	0.598581	0.550192	0.495056	0.495289
2	0.590555	0.585204	0.585786	0.562405	0.503780	0.502734
3	0.593347	0.593695	0.590090	0.557171	0.506805	0.505642
4	0.604397	0.596836	0.593695	0.559963	0.500989	0.500058
5	0.601954	0.596022	0.588112	0.553100	0.502850	0.501803
6	0.601838	0.597999	0.597650	0.553798	0.504246	0.503082
7	0.597185	0.595208	0.583343	0.550541	0.491451	0.491451
8	0.607189	0.604862	0.598348	0.565313	0.506688	0.504944
9	0.600093	0.594859	0.586484	0.555543	0.495405	0.494242
10	0.609980	0.601489	0.596255	0.558102	0.503548	0.502850
avg	0.601186	0.596289	0.591834	0.556613	0.501082	0.500209
std	0.006088	0.005130	0.005764	0.004972	0.005291	0.004860

Table 23: Label Propagation Accuracy

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.604048	0.596720	0.599860	0.569268	0.528324	0.518088
2	0.591951	0.586600	0.584739	0.576015	0.531232	0.523322
3	0.591834	0.590555	0.588577	0.567291	0.526114	0.522973
4	0.603583	0.595557	0.593114	0.575899	0.534372	0.530534
5	0.600093	0.592998	0.587763	0.575550	0.532279	0.515529
6	0.601605	0.598814	0.597534	0.574503	0.541352	0.527393
7	0.597069	0.598697	0.584623	0.573107	0.537280	0.517622
8	0.605444	0.602419	0.596603	0.582529	0.529254	0.530883
9	0.599744	0.595557	0.587182	0.577294	0.532279	0.536001
10	0.610097	0.602187	0.593928	0.575084	0.536583	0.523555
avg	0.600547	0.596010	0.591392	0.574654	0.532907	0.524590
std	0.005778	0.004973	0.005527	0.004203	0.004597	0.006593

Table 24: Label Spreading Accuracy

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.685006	0.680935	0.677911	0.646039	0.556357	0.536234
2	0.700942	0.691637	0.681400	0.658951	0.553682	0.541468
3	0.694079	0.684192	0.678492	0.649064	0.556124	0.531697
4	0.697453	0.690473	0.682447	0.655112	0.560777	0.546586
5	0.695243	0.683960	0.678841	0.650576	0.552984	0.542050
6	0.688612	0.692916	0.693149	0.649529	0.555426	0.542166
7	0.691055	0.688031	0.683262	0.641038	0.560893	0.544725
8	0.694894	0.688263	0.684541	0.651274	0.547284	0.541701
9	0.693847	0.691055	0.679888	0.646039	0.549610	0.544958
10	0.687449	0.679307	0.679074	0.648017	0.548680	0.529022
avg	0.692858	0.687077	0.681901	0.649564	0.554182	0.540060
std	0.004593	0.004722	0.004532	0.004978	0.004703	0.005848

Table 25: Wrapper Accuracy

7.2.2 Full Sensitivity Tables

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.689631	0.674885	0.679493	0.428802	0.002535	0.001152
2	0.672357	0.653648	0.665575	0.457203	0.003274	0.001169
3	0.674971	0.668625	0.666745	0.434313	0.005170	0.001645
4	0.692254	0.676204	0.675506	0.441498	0.003722	0.000233
5	0.689486	0.678271	0.668925	0.414953	0.004206	0.000234
6	0.686798	0.676030	0.674860	0.430243	0.004448	0.000468
7	0.676343	0.667429	0.656914	0.441829	0.003200	0.001143
8	0.691639	0.684829	0.681071	0.458196	0.004462	0.001174
9	0.685130	0.672719	0.666054	0.440588	0.004367	0.000689
10	0.691049	0.670951	0.671185	0.437018	0.005141	0.001402
avg	0.684966	0.672359	0.670633	0.438464	0.004052	0.000931
std	0.007553	0.008270	0.007319	0.012904	0.000858	0.000493

Table 26: Label Propagation Sensitivity

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.689862	0.688940	0.692627	0.666590	0.589631	0.547235
2	0.673994	0.673293	0.674930	0.669083	0.585594	0.548644
3	0.675441	0.682961	0.677321	0.657109	0.580024	0.546181
4	0.694348	0.690858	0.688997	0.680391	0.595720	0.557106
5	0.690187	0.689953	0.680607	0.668458	0.597897	0.556075
6	0.689139	0.690075	0.685861	0.663155	0.604401	0.548221
7	0.678857	0.686171	0.670171	0.665600	0.594743	0.550857
8	0.692579	0.698215	0.691404	0.678957	0.598168	0.553311
9	0.686739	0.687658	0.679614	0.681223	0.591358	0.565157
10	0.694555	0.690582	0.680533	0.658331	0.601075	0.555971
avg	0.686570	0.687871	0.682207	0.668890	0.593861	0.552876
std	0.007691	0.006439	0.007351	0.008725	0.007350	0.005860

Table 27: Label Spreading Sensitivity

Fold	0.1	0.2	0.5	0.9	0.95
1	0.652535	0.673502	0.735023	0.704378	0.630876
2	0.654116	0.666277	0.734097	0.675398	0.636342
3	0.654994	0.670035	0.721034	0.677791	0.634313
4	0.656199	0.678297	0.740870	0.686904	0.619214
5	0.664720	0.672196	0.724766	0.687850	0.648364
6	0.661049	0.692884	0.737594	0.679073	0.617509
7	0.659657	0.679771	0.725257	0.658971	0.657829
8	0.665336	0.674730	0.740019	0.688351	0.642790
9	0.667203	0.677545	0.732016	0.668812	0.634567
10	0.642206	0.667679	0.717224	0.670951	0.633092
avg	0.657801	0.675292	0.730790	0.679848	0.635490
std	0.007468	0.007628	0.008230	0.012690	0.012185

Table 28: Wrapper Sensitivity

7.2.3 Full Specificity Tables

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.519380	0.517031	0.516091	0.673949	0.997181	0.999060
2	0.509604	0.517473	0.506827	0.666512	0.999074	0.999074
3	0.513358	0.520267	0.514970	0.677568	0.998388	0.999539
4	0.516519	0.517450	0.511866	0.678455	0.998371	1.000000
5	0.515173	0.514478	0.507992	0.690063	0.997220	0.999073
6	0.517919	0.520925	0.521387	0.675838	0.997919	0.999538
7	0.515159	0.520369	0.507106	0.663193	0.997395	0.999526
8	0.524314	0.526389	0.517170	0.670431	0.999539	0.999309
9	0.512953	0.515073	0.504946	0.673340	0.998587	1.000000
10	0.529643	0.532654	0.522001	0.678092	0.997453	0.999768
avg	0.517402	0.520211	0.513036	0.674744	0.998113	0.999489
std	0.005865	0.005558	0.006203	0.007388	0.000815	0.000360

Table 29: Label Propagation Specificity

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.516561	0.502701	0.505285	0.470049	0.465821	0.488372
2	0.510761	0.500810	0.495487	0.483916	0.477436	0.498264
3	0.509903	0.500000	0.501612	0.479272	0.473284	0.500230
4	0.512797	0.500233	0.497208	0.471382	0.473011	0.503955
5	0.510771	0.496873	0.495715	0.483438	0.467223	0.475330
6	0.515145	0.508671	0.510289	0.486936	0.479075	0.506821
7	0.512316	0.508053	0.495973	0.477262	0.477736	0.483183
8	0.519935	0.508412	0.503572	0.487900	0.461627	0.508873
9	0.510598	0.501178	0.492463	0.470796	0.471738	0.506123
10	0.526401	0.514590	0.508106	0.492589	0.472673	0.491431
avg	0.514519	0.504152	0.500571	0.480354	0.471962	0.496258
std	0.005247	0.005482	0.006066	0.007895	0.005620	0.011262

Table 30: Label Spreading Specificity

Fold	0.1	0.2	0.5	0.9	0.95
1	0.709890	0.682405	0.555321	0.405450	0.439746
2	0.728766	0.696367	0.584587	0.433233	0.447582
3	0.712805	0.686780	0.578535	0.436895	0.431138
4	0.724756	0.686598	0.569335	0.434621	0.473941
5	0.703035	0.685430	0.577021	0.419273	0.436646
6	0.724393	0.693410	0.562543	0.433295	0.467746
7	0.717432	0.686878	0.553766	0.459261	0.427523
8	0.710763	0.694169	0.564185	0.408850	0.442498
9	0.715497	0.682289	0.557937	0.427461	0.453132
10	0.716072	0.690366	0.579435	0.427513	0.425892
avg	0.716341	0.688469	0.568266	0.428585	0.444584
std	0.007848	0.004898	0.011097	0.015290	0.016304

Table 31: Wrapper Specificity

7.2.4 Full F1 Score Table

Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.638234	0.628204	0.630870	0.490447	0.005042	0.002299
2	0.620280	0.610529	0.615152	0.509645	0.006521	0.002334
3	0.621645	0.619623	0.616873	0.492603	0.010271	0.003283
4	0.636373	0.626509	0.624449	0.500858	0.007404	0.000465
5	0.632990	0.625714	0.617891	0.480390	0.008353	0.000467
6	0.631579	0.625650	0.625041	0.489350	0.008837	0.000935
7	0.630850	0.626609	0.616077	0.500129	0.006364	0.002282
8	0.635589	0.631921	0.626824	0.510800	0.008881	0.002344
9	0.634255	0.626968	0.619827	0.500849	0.008684	0.001378
10	0.638178	0.626309	0.623332	0.496087	0.010204	0.002800
avg	0.631997	0.624804	0.621633	0.497116	0.008056	0.001859
std	0.006335	0.005845	0.005244	0.009353	0.001700	0.000984

Table 32: Label Propagation F1 Score

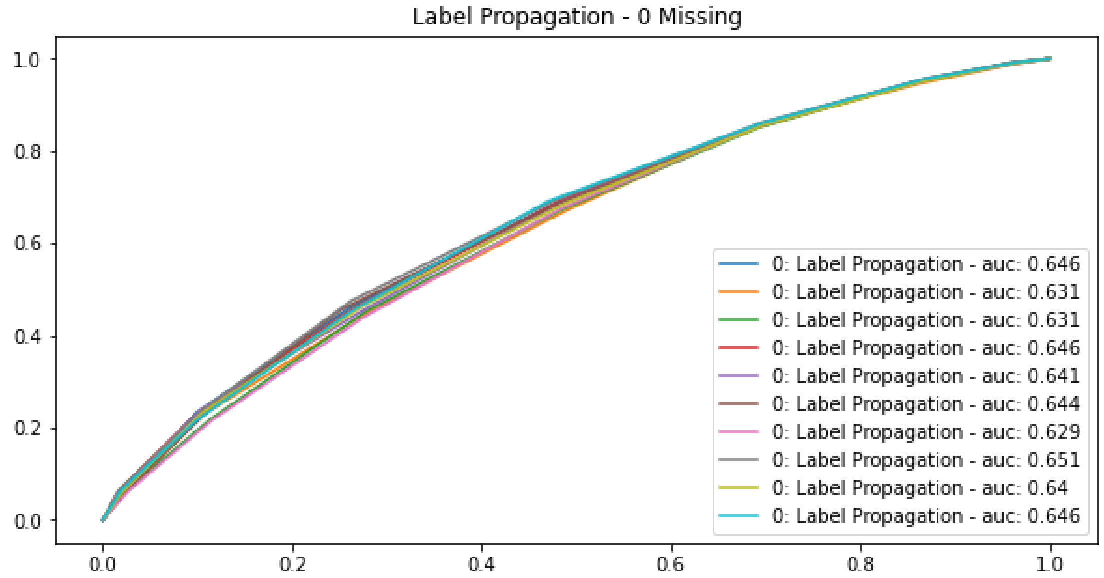
Fold	0	0.1	0.2	0.5	0.9	0.95
1	0.637564	0.633005	0.636056	0.609759	0.557942	0.534128
2	0.621657	0.618342	0.617855	0.610868	0.554105	0.533788
3	0.620936	0.622803	0.619718	0.600515	0.547836	0.531261
4	0.636596	0.630774	0.628741	0.616049	0.561315	0.542715
5	0.632142	0.627964	0.621772	0.610607	0.560018	0.533333
6	0.632235	0.630926	0.628755	0.607679	0.567036	0.535498
7	0.631646	0.635075	0.621516	0.613440	0.566761	0.537526
8	0.634876	0.634985	0.629329	0.617010	0.557269	0.538822
9	0.634597	0.632491	0.624960	0.619954	0.561361	0.552150
10	0.639415	0.633441	0.625228	0.606654	0.563541	0.537384
avg	0.632166	0.629981	0.625393	0.611254	0.559718	0.537661
std	0.006249	0.005487	0.005468	0.005663	0.005821	0.006044

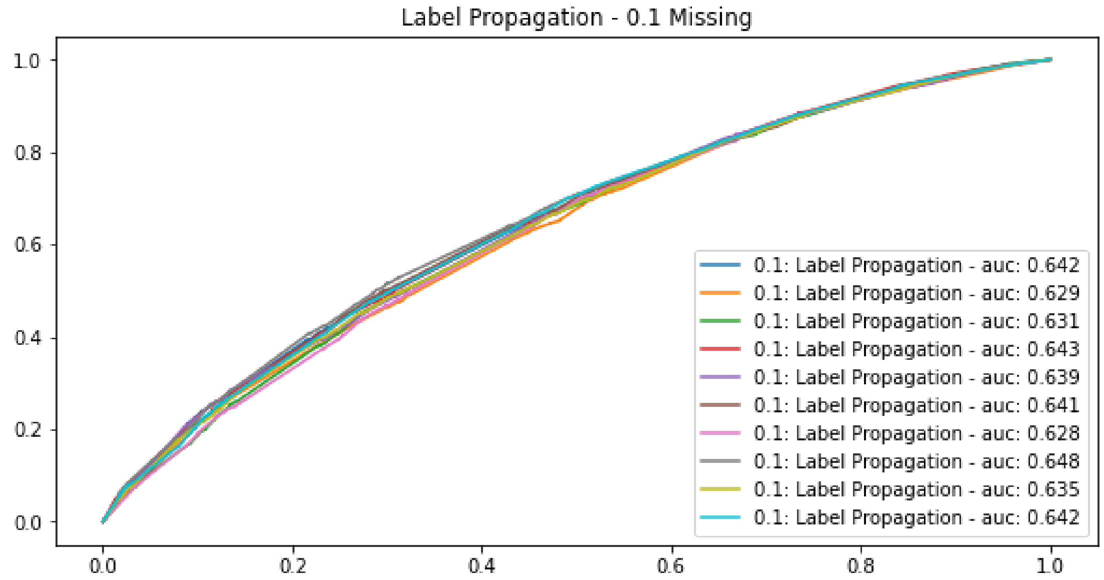
Table 33: Label Spreading F1 Score

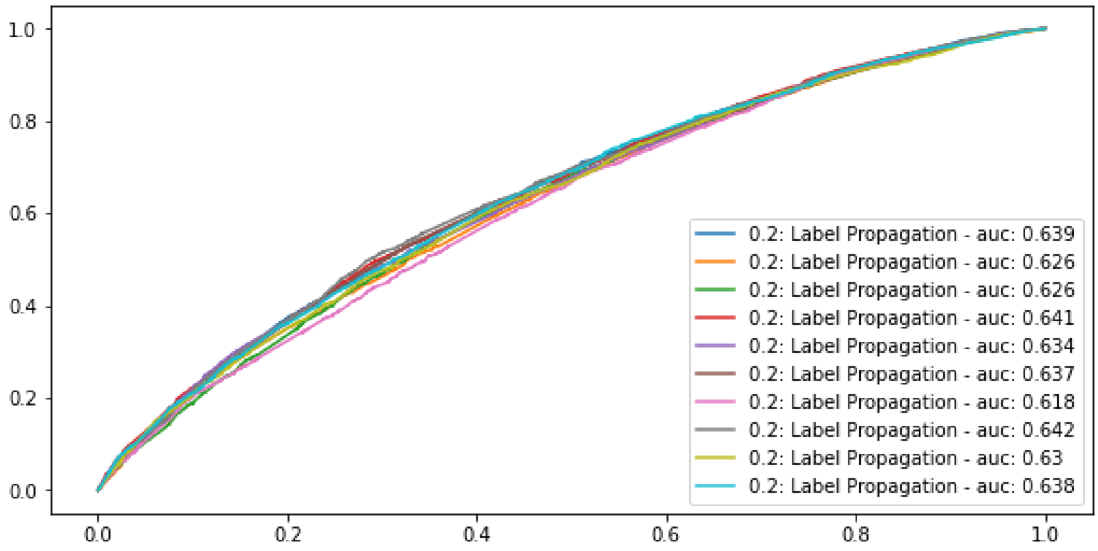
Fold	0.1	0.2	0.5	0.9	0.95
1	0.673724	0.678584	0.677067	0.615834	0.578675
2	0.678472	0.675359	0.681650	0.600853	0.579923
3	0.672457	0.673518	0.670381	0.601836	0.572793
4	0.679513	0.681149	0.682378	0.609998	0.577315
5	0.676817	0.675749	0.673762	0.605077	0.585011
6	0.681467	0.691750	0.676543	0.602868	0.572731
7	0.682754	0.685965	0.672816	0.604339	0.595243
8	0.678888	0.679357	0.677634	0.600984	0.581474
9	0.686126	0.681776	0.676724	0.600495	0.585330
10	0.665940	0.674377	0.669795	0.596757	0.572304
avg	0.677616	0.679758	0.675875	0.603904	0.580080
std	0.005757	0.005708	0.004247	0.005446	0.007157

Table 34: Wrapper F1 Score

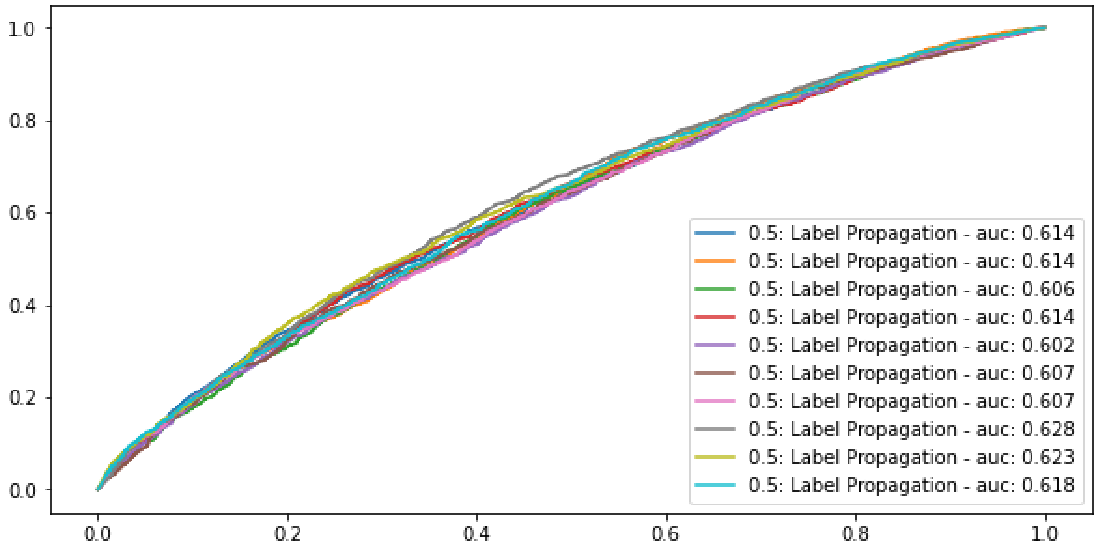
7.2.5 ROC Curves



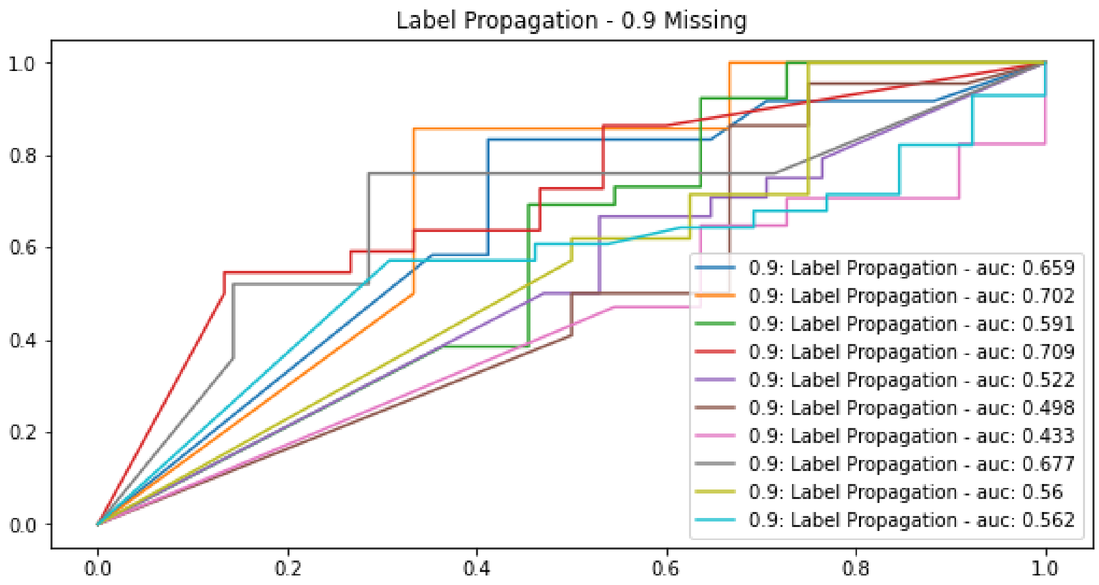


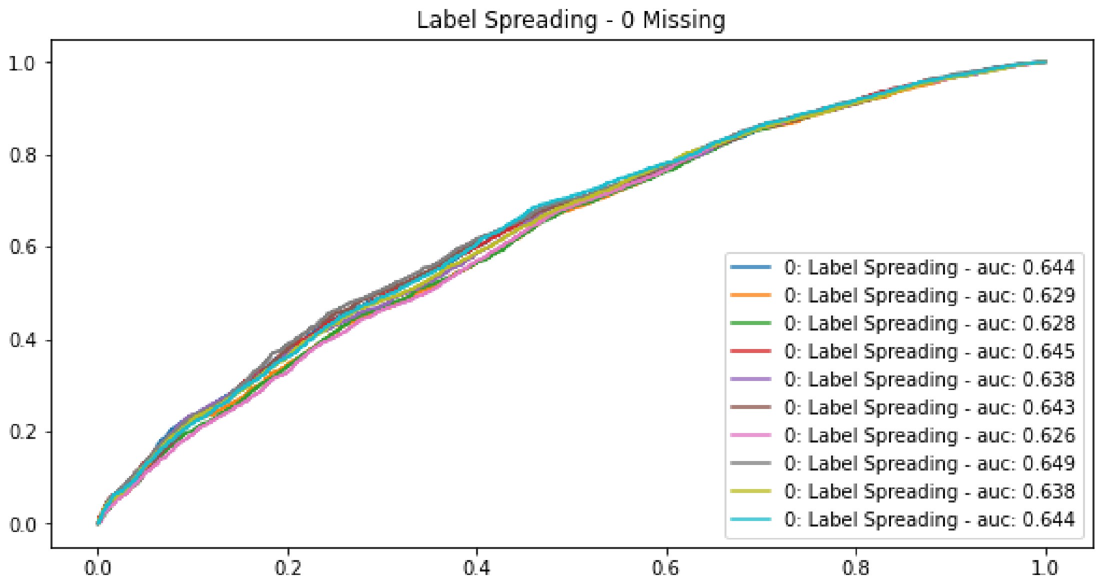


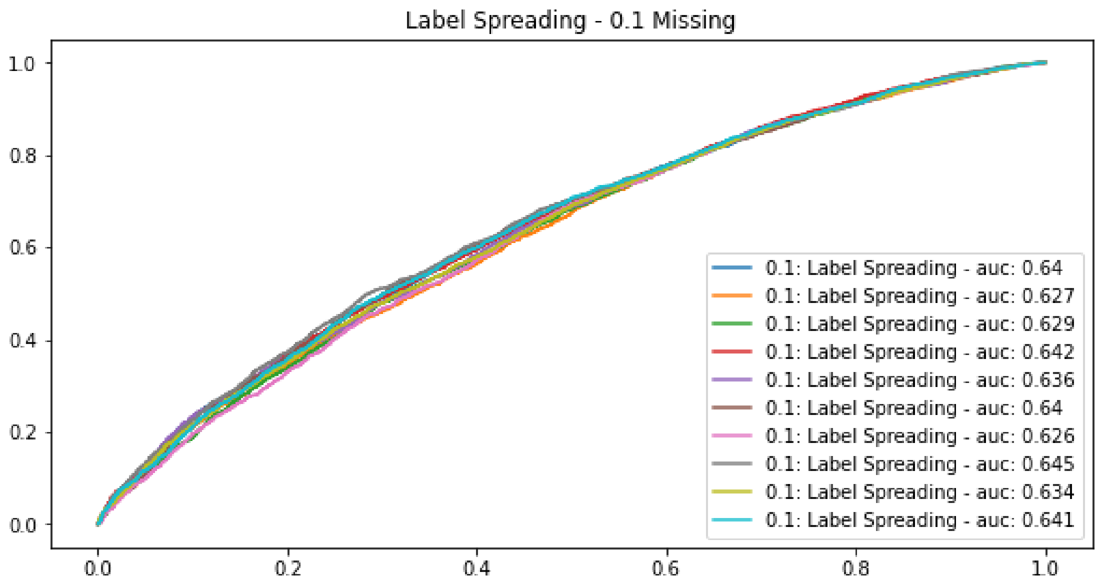
Label Propagation - 0.2 Missing

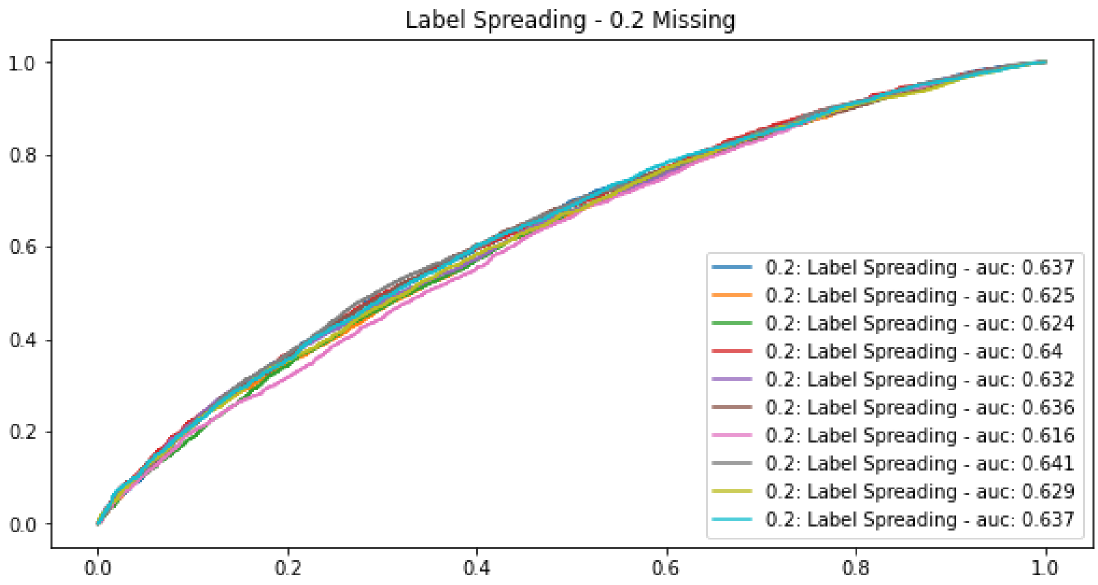


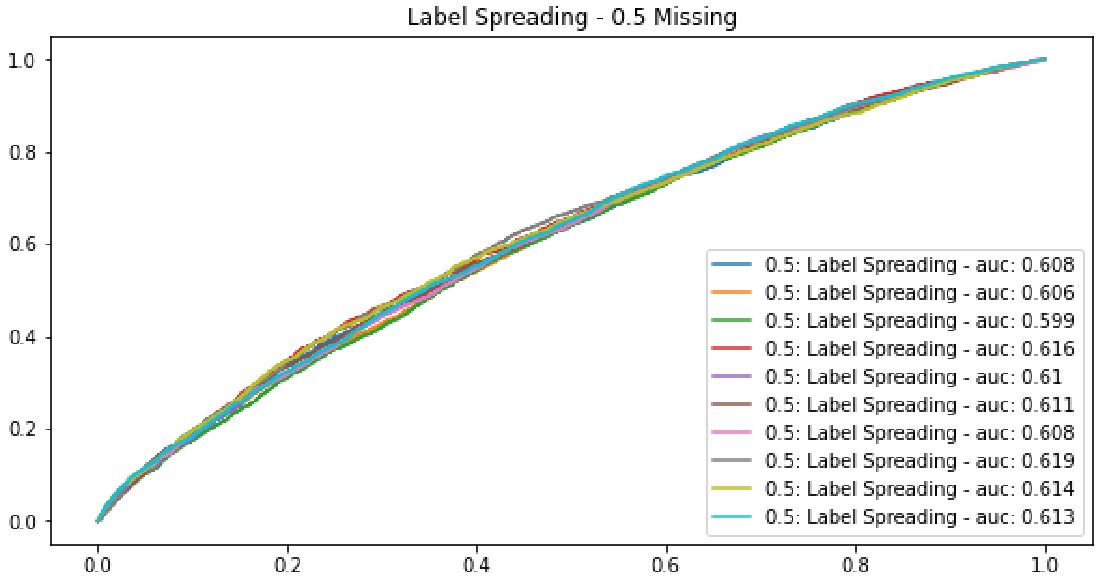
Label Propagation - 0.5 Missing

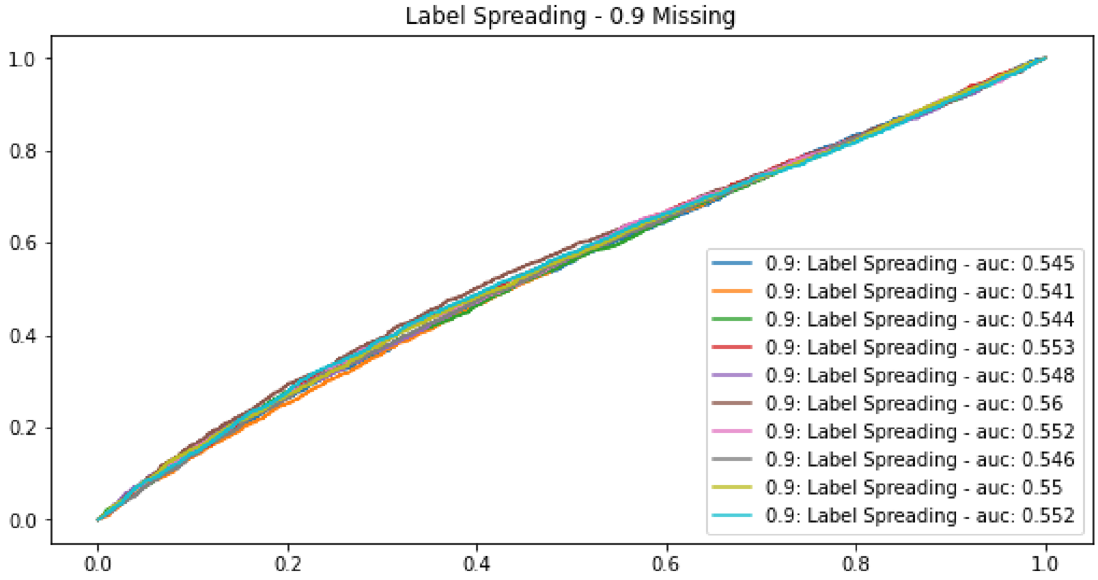

Label Propagation - 0.9 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.9: Label Propagation - auc: 0.659
0.9: Label Propagation - auc: 0.702
0.9: Label Propagation - auc: 0.591
0.9: Label Propagation - auc: 0.709
0.9: Label Propagation - auc: 0.522
0.9: Label Propagation - auc: 0.498
0.9: Label Propagation - auc: 0.433
0.9: Label Propagation - auc: 0.677
0.9: Label Propagation - auc: 0.56
0.9: Label Propagation - auc: 0.562

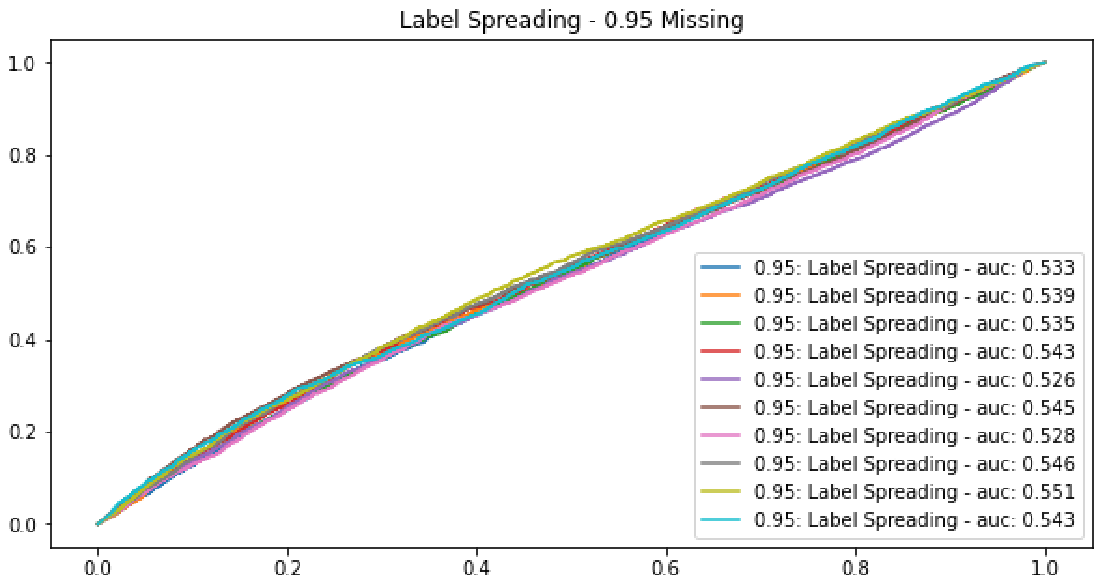

Label Spreading - 0 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0: Label Spreading - auc: 0.644
0: Label Spreading - auc: 0.629
0: Label Spreading - auc: 0.628
0: Label Spreading - auc: 0.645
0: Label Spreading - auc: 0.638
0: Label Spreading - auc: 0.643
0: Label Spreading - auc: 0.626
0: Label Spreading - auc: 0.649
0: Label Spreading - auc: 0.638
0: Label Spreading - auc: 0.644

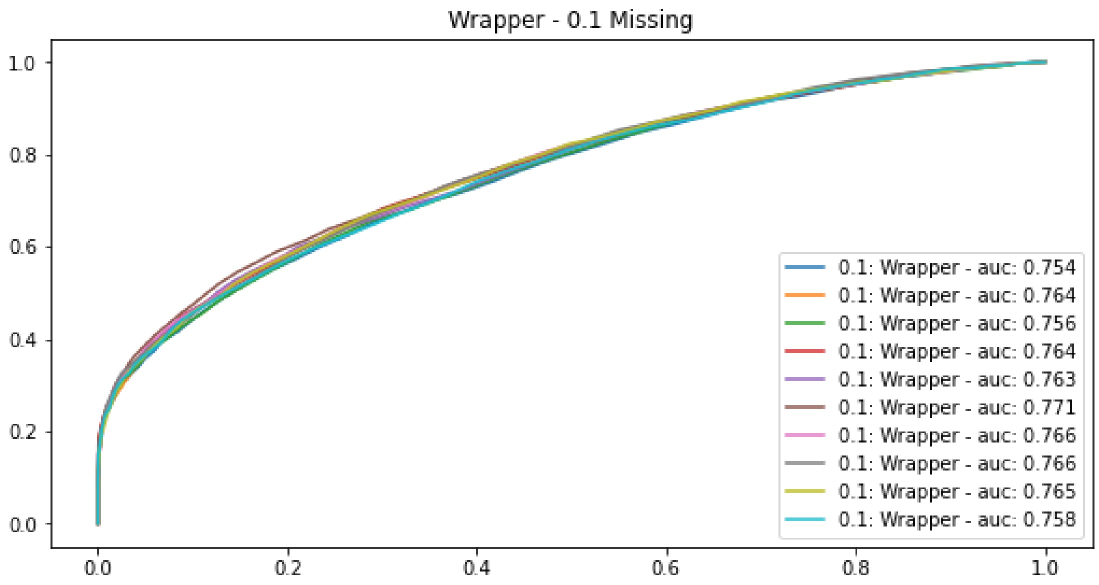

Label Spreading - 0.1 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.1: Label Spreading - auc: 0.64
0.1: Label Spreading - auc: 0.627
0.1: Label Spreading - auc: 0.629
0.1: Label Spreading - auc: 0.642
0.1: Label Spreading - auc: 0.636
0.1: Label Spreading - auc: 0.64
0.1: Label Spreading - auc: 0.626
0.1: Label Spreading - auc: 0.645
0.1: Label Spreading - auc: 0.634
0.1: Label Spreading - auc: 0.641

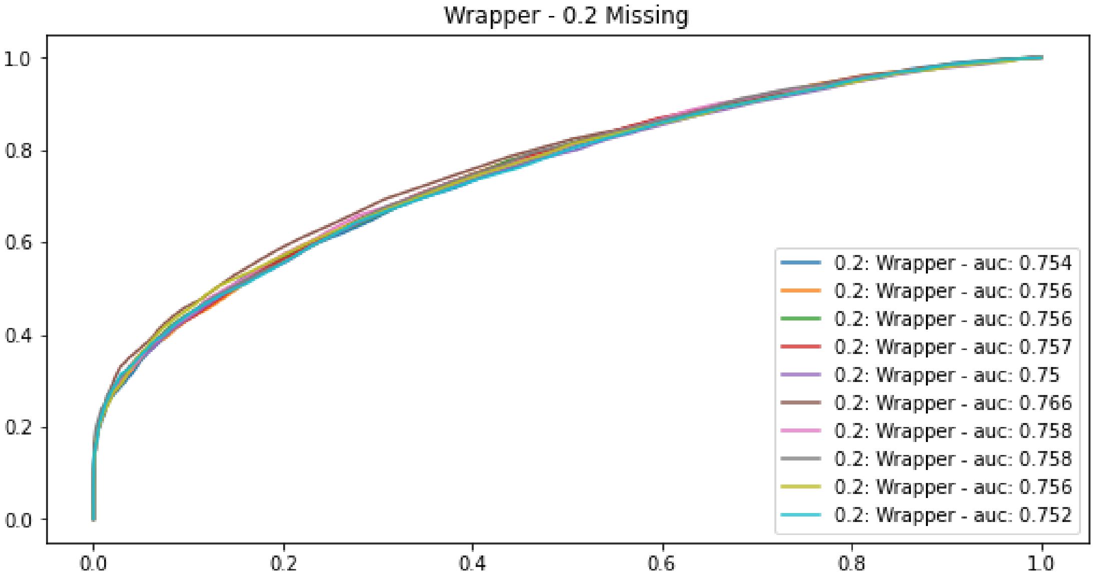

Label Spreading - 0.2 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2: Label Spreading - auc: 0.637
0.2: Label Spreading - auc: 0.625
0.2: Label Spreading - auc: 0.624
0.2: Label Spreading - auc: 0.64
0.2: Label Spreading - auc: 0.632
0.2: Label Spreading - auc: 0.636
0.2: Label Spreading - auc: 0.616
0.2: Label Spreading - auc: 0.641
0.2: Label Spreading - auc: 0.629
0.2: Label Spreading - auc: 0.637

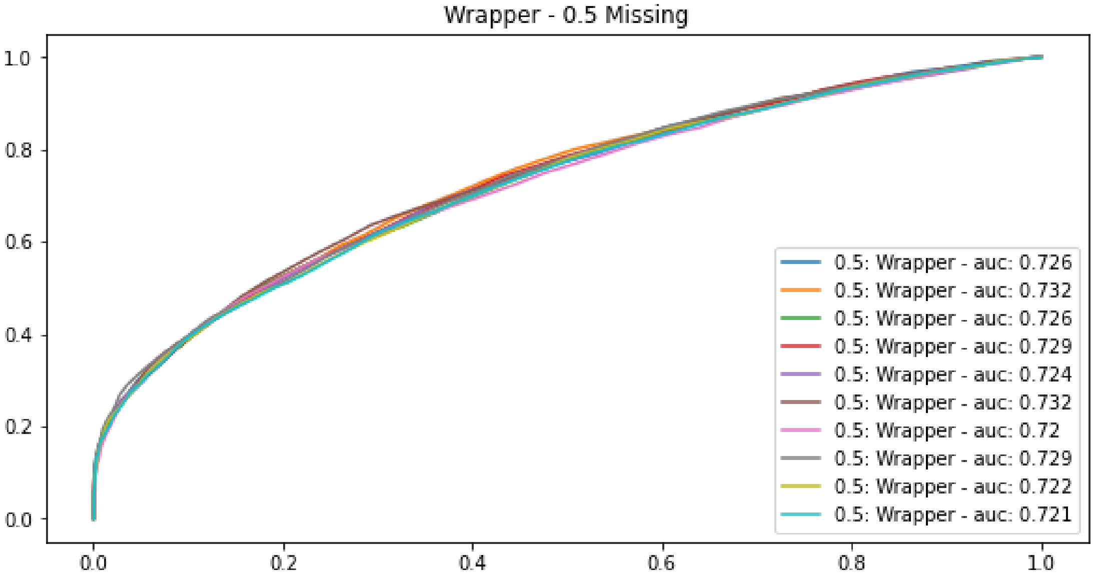

Label Spreading - 0.5 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.5: Label Spreading - auc: 0.608
0.5: Label Spreading - auc: 0.606
0.5: Label Spreading - auc: 0.599
0.5: Label Spreading - auc: 0.616
0.5: Label Spreading - auc: 0.61
0.5: Label Spreading - auc: 0.611
0.5: Label Spreading - auc: 0.608
0.5: Label Spreading - auc: 0.619
0.5: Label Spreading - auc: 0.614
0.5: Label Spreading - auc: 0.613

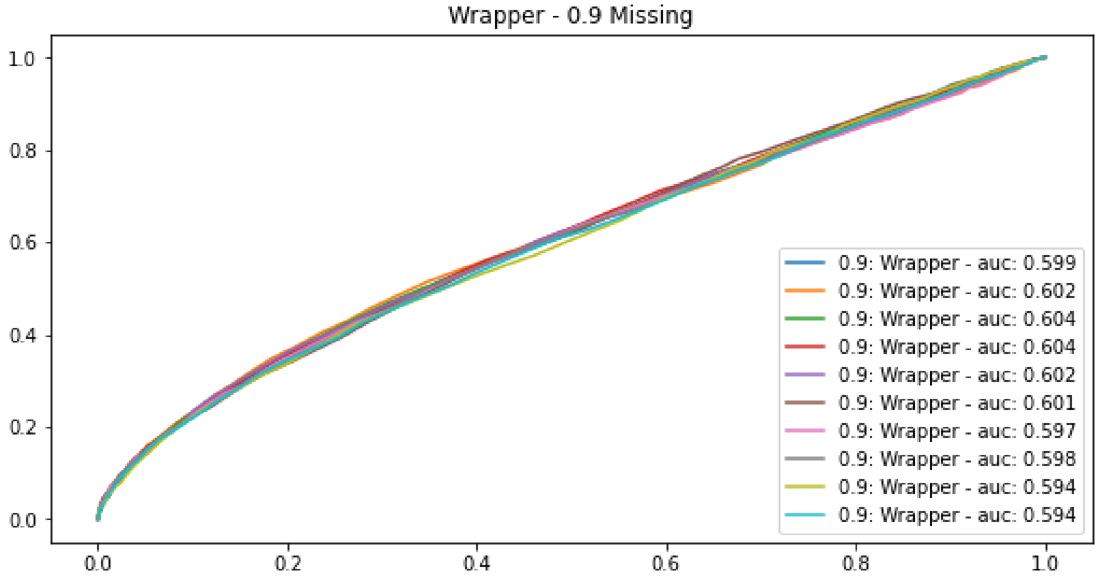

Label Spreading - 0.9 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.9: Label Spreading - auc: 0.545
0.9: Label Spreading - auc: 0.541
0.9: Label Spreading - auc: 0.544
0.9: Label Spreading - auc: 0.553
0.9: Label Spreading - auc: 0.548
0.9: Label Spreading - auc: 0.56
0.9: Label Spreading - auc: 0.552
0.9: Label Spreading - auc: 0.546
0.9: Label Spreading - auc: 0.55
0.9: Label Spreading - auc: 0.552

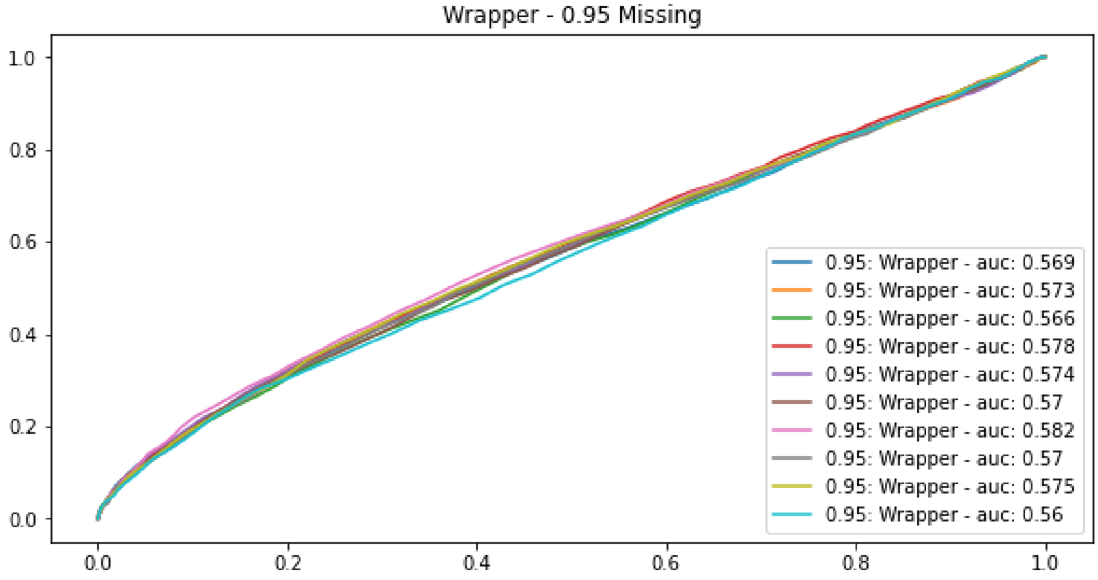

Label Spreading - 0.95 Missing
0.95: Label Spreading - auc: 0.533
0.95: Label Spreading - auc: 0.539
0.95: Label Spreading - auc: 0.535
0.95: Label Spreading - auc: 0.543
0.95: Label Spreading - auc: 0.526
0.95: Label Spreading - auc: 0.545
0.95: Label Spreading - auc: 0.528
0.95: Label Spreading - auc: 0.546
0.95: Label Spreading - auc: 0.551
0.95: Label Spreading - auc: 0.543
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Wrapper - 0.1 Missing
0.1: Wrapper - auc: 0.754
0.1: Wrapper - auc: 0.764
0.1: Wrapper - auc: 0.756
0.1: Wrapper - auc: 0.764
0.1: Wrapper - auc: 0.763
0.1: Wrapper - auc: 0.771
0.1: Wrapper - auc: 0.766
0.1: Wrapper - auc: 0.766
0.1: Wrapper - auc: 0.765
0.1: Wrapper - auc: 0.758
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Wrapper - 0.2 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2: Wrapper - auc: 0.754
0.2: Wrapper - auc: 0.756
0.2: Wrapper - auc: 0.756
0.2: Wrapper - auc: 0.757
0.2: Wrapper - auc: 0.75
0.2: Wrapper - auc: 0.766
0.2: Wrapper - auc: 0.758
0.2: Wrapper - auc: 0.758
0.2: Wrapper - auc: 0.756
0.2: Wrapper - auc: 0.752


Wrapper - 0.5 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.5: Wrapper - auc: 0.726
0.5: Wrapper - auc: 0.732
0.5: Wrapper - auc: 0.726
0.5: Wrapper - auc: 0.729
0.5: Wrapper - auc: 0.724
0.5: Wrapper - auc: 0.732
0.5: Wrapper - auc: 0.72
0.5: Wrapper - auc: 0.729
0.5: Wrapper - auc: 0.722
0.5: Wrapper - auc: 0.721


Wrapper - 0.9 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.9: Wrapper - auc: 0.599
0.9: Wrapper - auc: 0.602
0.9: Wrapper - auc: 0.604
0.9: Wrapper - auc: 0.604
0.9: Wrapper - auc: 0.602
0.9: Wrapper - auc: 0.601
0.9: Wrapper - auc: 0.597
0.9: Wrapper - auc: 0.598
0.9: Wrapper - auc: 0.594
0.9: Wrapper - auc: 0.594


Wrapper - 0.95 Missing
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.95: Wrapper - auc: 0.569
0.95: Wrapper - auc: 0.573
0.95: Wrapper - auc: 0.566
0.95: Wrapper - auc: 0.578
0.95: Wrapper - auc: 0.574
0.95: Wrapper - auc: 0.57
0.95: Wrapper - auc: 0.582
0.95: Wrapper - auc: 0.57
0.95: Wrapper - auc: 0.575
0.95: Wrapper - auc: 0.56