Credit card fraud detection

Niels Krog - Bjarne Munch
June 2021

All group members have contributed equally to this project.

Outline

- The problem
- The dataset
- Data preprocessing
- The models
- Training and evaluation
- Results
- Discussion
- Conclusion

The Problem

Credit card information is leaked and is used by thieves.

You should not be charged with their expenses.

According to European Central Bank, €1.8 billion was lost due to credit card fraud in 2018 in Europe alone [1]

Automatic credit card fraud detection must be fast and reliable, so the fraud is detected before serious damage happens.

As fraud detection techniques evolve, so do the tactics used by the fraudsters, creating a cat and mouse game[1]

1 - Borgne, Yann-Aël Le, and Gianluca Bontempi. “Machine Learning for Credit Card Fraud Detection - Practical Handbook.” *Credit Card Fraud Scenarios*, 2021, fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_2_Background/CreditCardFraud.html.

About the Dataset

Using dataset from kaggle[1]

Credit card transactions from 2013 by European card holders.

Highly imbalanced with 284,807 records of which 492 are fraudulent (ie. 0.172%)

26 numerical features obtained by PCA reduction to anonymize data.

2 raw numerical features: time between transactions and amount spent.

Binary target value.

1 - https://www.kaggle.com/mlg-ulb/creditcardfraud

Classification challenges

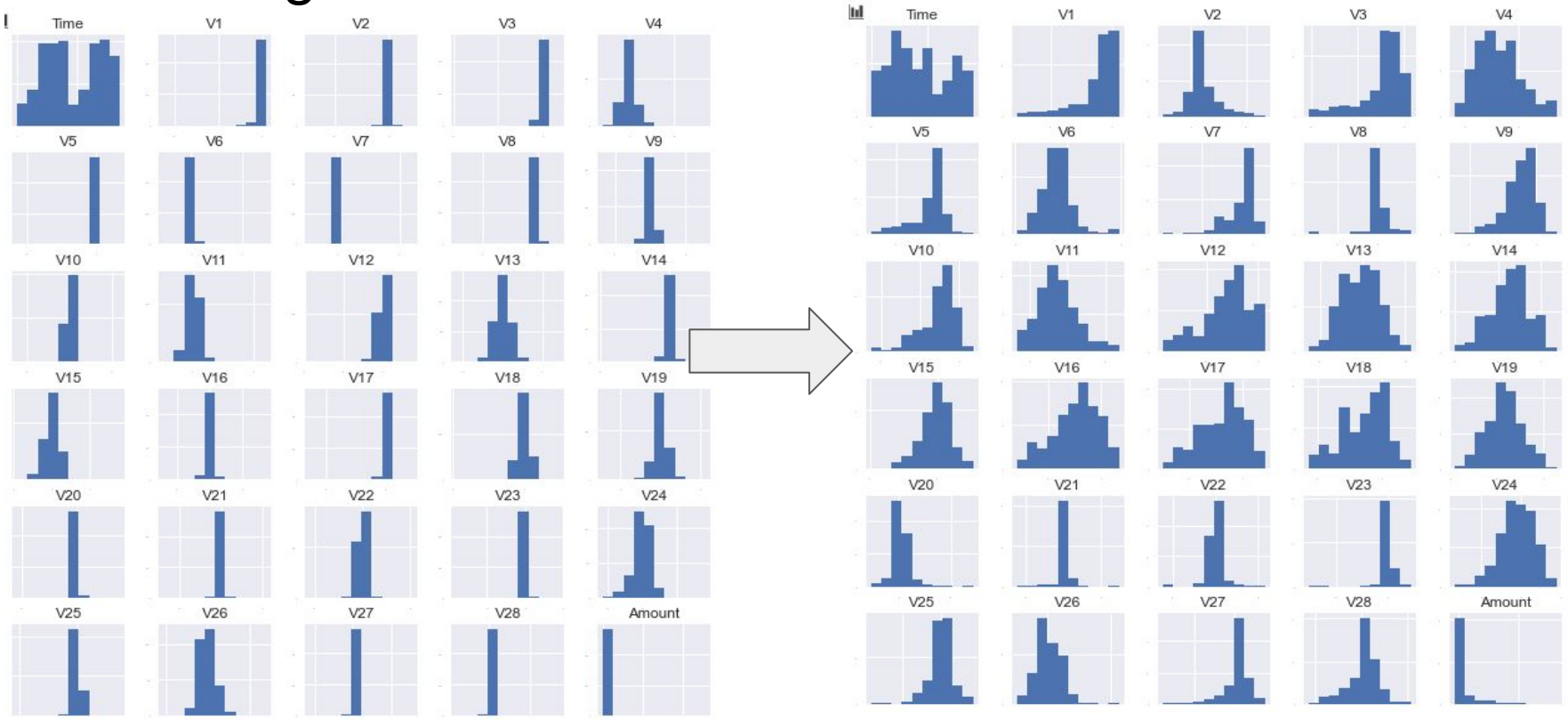
The vast majority of transactions are not fraudulent, making the two categories highly imbalanced.

A baseline classifier would often get 99% accuracy, therefore we need a better performance metric[1]

False negatives are costly and false positives cause customer dissatisfaction.

1 - Borgne, Yann-Aël Le, and Gianluca Bontempi. “Machine Learning for Credit Card Fraud Detection - Practical Handbook.” *Introduction*, 2021, https://fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_4_PerformanceMetrics/Introduction.html

Normalizing the data set

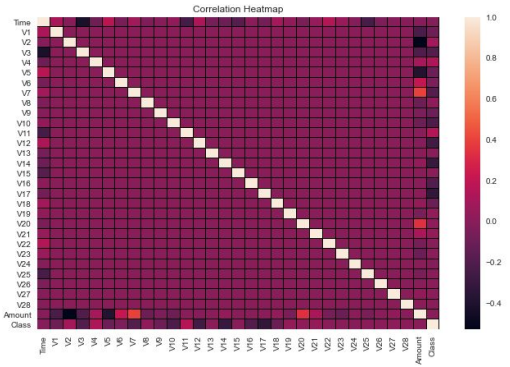


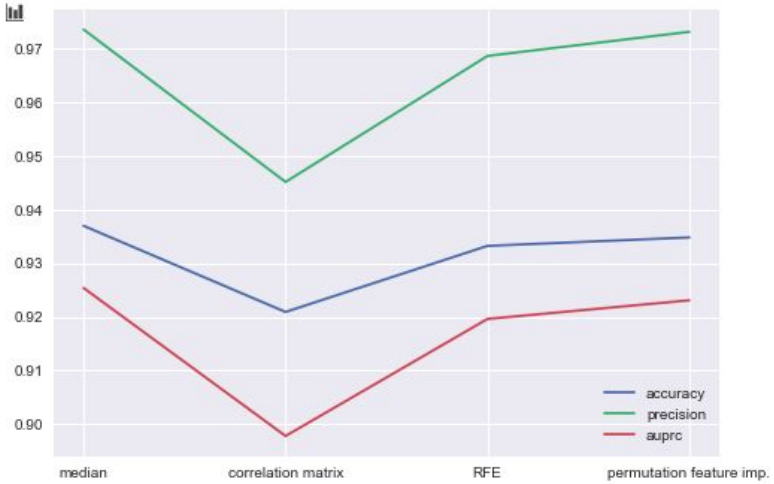
Feature importance

We experimented with different strategies for finding the best features:

1. Largest difference in median - `['V3', 'V4', 'V7', 'V10', 'V11', 'V12', 'V14', 'V16', 'V17', 'V18']`
2. Correlation Heatmap - `['V11','V4','V2','V21','V19','V20','V8','V27','V28']`
3. Recursive Feature Elimination - `['V4', 'V8', 'V9', 'V10', 'V13', 'V14', 'V16', 'V21', 'V22', 'V27']`
4. Permutation Feature Importance - `['V14','V12','V17','V10','V4','V16','V7','V2','V11','Amount']`

Using this, we found (1) to yield the best result



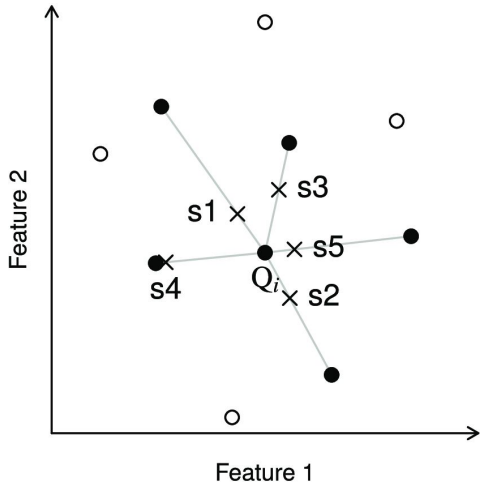


Over- / undersampling / SMOTE

To compensate for imbalance in data, one can:

- **Under-sample**: Remove samples from the over-represented class. This is mostly done to save space or processing time, as it may discard important information.

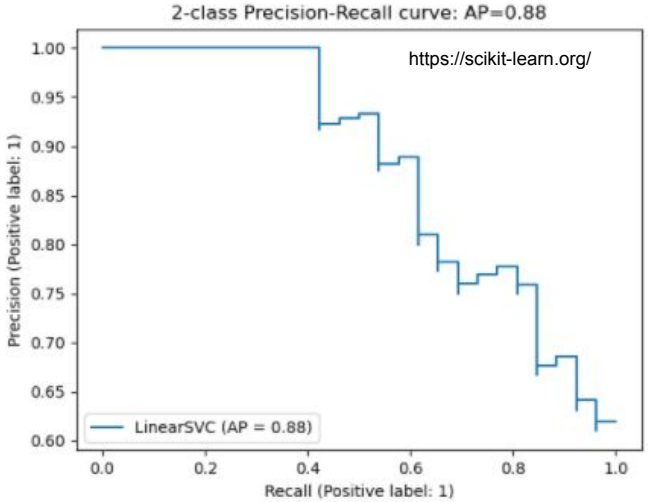
- **Over-sample:** Randomly add copies of the under-represented class to the data. This is not suitable for training, where you may transfer knowledge in the copied samples.
  SMOTE is an over-sampling technique, where synthetic samples are generated using the k-nearest neighbors.



https://www.researchgate.net/figure/Illustration-of-SMOTE-Synthetic-points-crosses-denoted-s1-through-s5-generated-by_fig1_322701982

Choosing an evaluation metric

Choosing the right metric for evaluating the classification, has great impact on which classifier is perceived to be best. We tried each of the following:



1. Accuracy - This is the default metric and is not suited for this problem.
2. False negatives, False positives, True positives - Gives insight in how it classifies.
3. F1 score - Is considered suited for imbalanced datasets.
4. AUC ROC - Area Under Curve Receiving Operating Characteristic is widely used for this specific task, but not good for imbalanced datasets
5. AUPRC - Area Under Precision-Recall Curve is the suggested metric from Kaggle.
6. Average Precision - Should be a good overall measure for this task.
7. Mean score for metrics - The mean for 3, 4, 5, 6

Eventually settling on (5) AUPRC, which seems most suited for highly imbalanced datasets.

In summary...

- Three lists of features, which each model was trained on.
- Training data was balanced using SMOTE - test data remained unaltered.
- Models were validated using stratified 5-fold cross validation (due to imbalanced data) - However, all results are from unaltered test data.

Classification Models

Tried different libraries

- Models from Scikit-Learn
- Keras Neural Network
- PyTorch Neural Network
- XGBoost

Scikit-Learn Models

- DummyClassifier as baseline
- Logistic Regression
- K-Nearest Neighbours (5 neighbours)
- Random Forest (300 estimators)
- Naive Bayes
- Multilayer Perceptron (1000, 500)

KNN and NB just included for comparison, do not expect to perform well.

Hyperparameters found manually

Randomized search would focus on one feature combination

XGBoost Model

Random Forest was performing very well. We added XGBoost for direct comparison to the Random Forest.

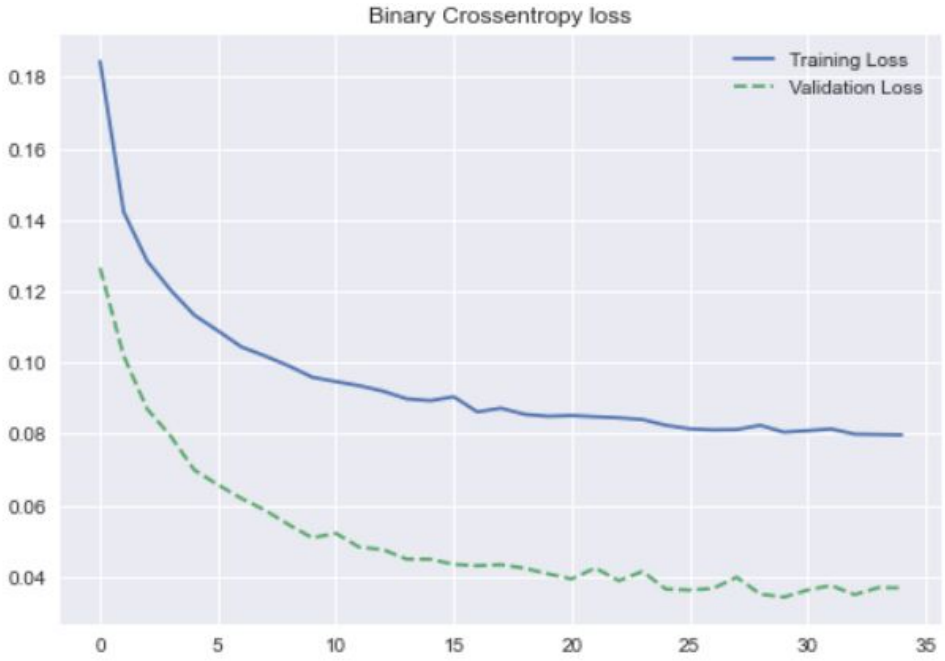
It also has 300 estimators.

Keras Sequential / PyTorch Neural Network

SMOTE improved the accuracy immensely!

3 hidden dense layers (256,128,64)

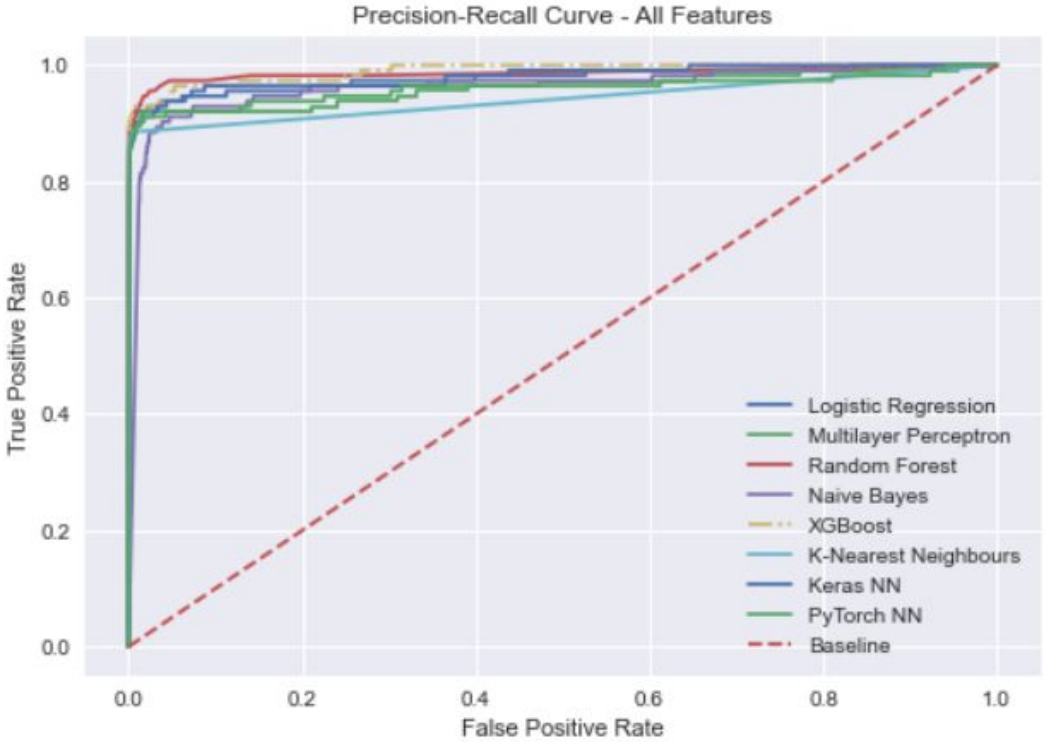
1 output layer sigmoid neuron

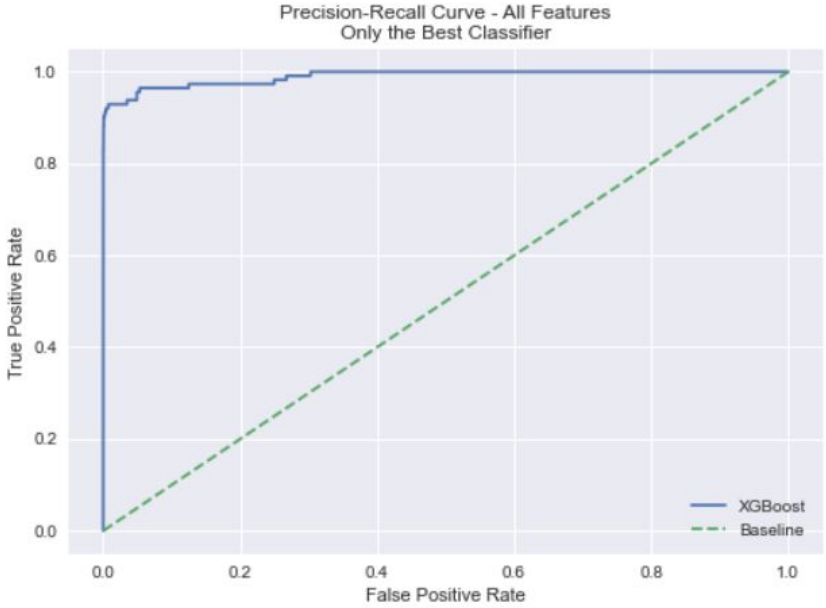


Results - AUPRC on test set

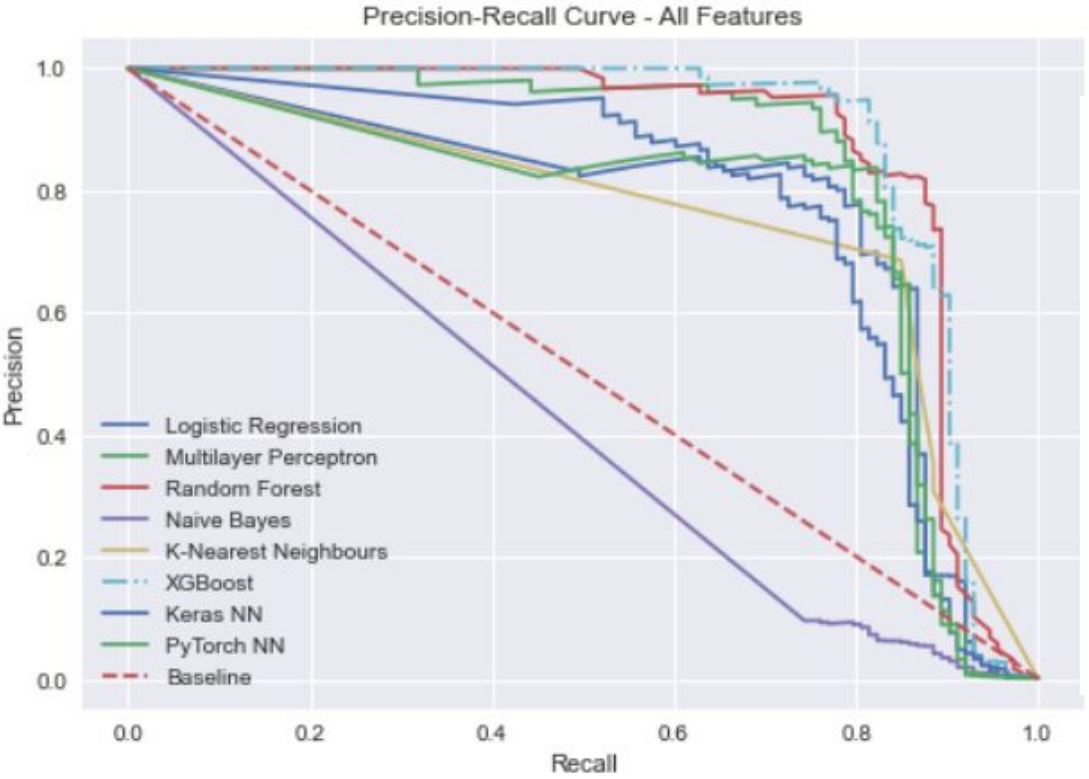
Model	All Features	Max Median	Permutation Import.
Baseline	0.50	0.50	0.50
Logistic Regression	0.76	0.73	0.73
Naive Bayes	0.42	0.53	0.51
K-Nearest Neighbours	0.75	0.67	0.64
Random Forest	0.87	**0.84**	0.84
Multilayer Perceptron	0.81	0.76	0.76
Keras NN	0.78	0.77	0.74
PyTorch NN	0.76	0.77	0.76
XGBoost	**0.88**	0.83	**0.85**

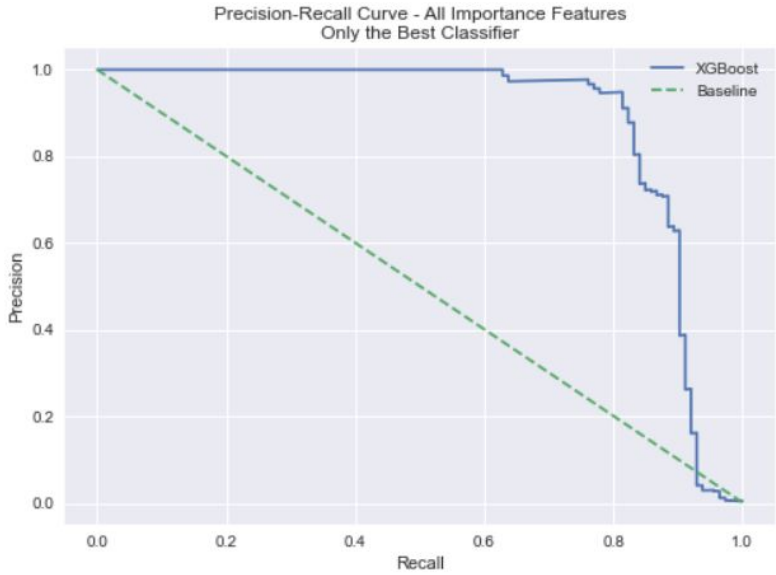
Results - ROC curve





Results - PR-Curve





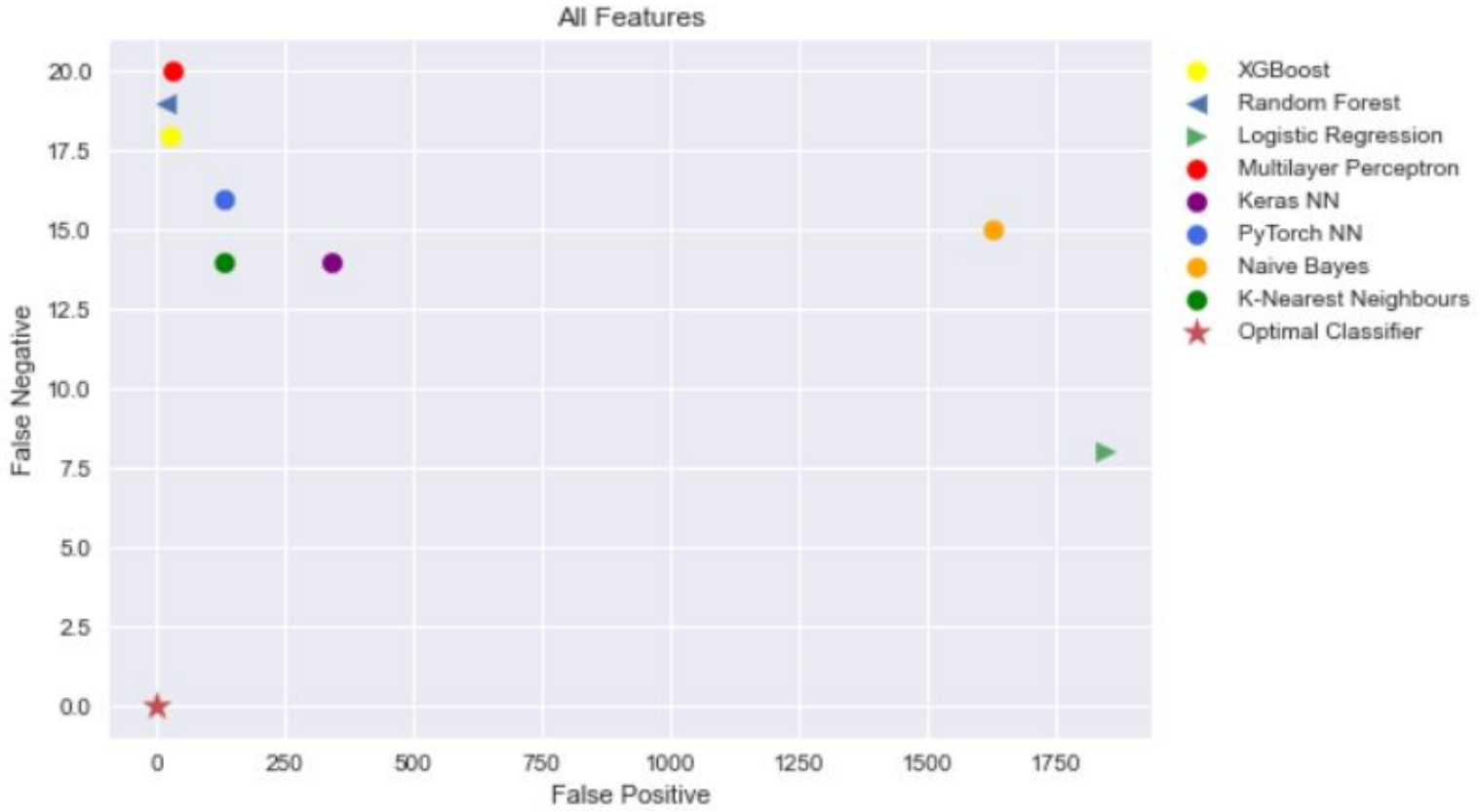
Results - False Negatives

Model	All Features	Max Median	Permutation Import.
Baseline	113	113	113
Logistic Regression	**8**	**10**	**10**
Naive Bayes	15	15	15
K-Nearest Neighbours	14	15	12
Random Forest	19	16	18
Multilayer Perceptron	18	16	16
Keras NN	14	11	11
PyTorch NN	11	12	11
XGBoost	18	16	16

Results - False Positives

Model	All Features	Max Median	Permutation Import.
Baseline	0	0	0
Logistic Regression	1844	1860	1873
Naive Bayes	1625	1026	1143
K-Nearest Neighbours	130	219	270
Random Forest	**19**	**36**	**33**
Multilayer Perceptron	31	52	75
Keras NN	339	1205	1747
PyTorch NN	62	52	63
XGBoost	27	72	58

Results - False Positives and Negatives



Can we Boost the Results?

Combining the predictions made by each classifier (excluding baseline and Naive Bayes)

	Threshold	AUPRC	FP	FN
All Features	0.89	0.87	**5**	23
Max Median	0.90	**0.86**	**8**	22
Permutation	0.86	0.85	**10**	22

Discussion

Seems like it is a tradeoff between false positives and negatives.

Naive Bayes is just overall bad. Likely due to feature independence.

K-Nearest Neighbours not too good either - Curse of dimensionality.

Ensemble classifiers are good as always!

Discussion - Neural Networks

How come Keras/PyTorch NN be worse than many other models?

- So many things to tune
- Long training time
- Would likely outperform many of them with a different architecture

Conclusion

Tree-based classifiers are the best solution!

They are easy to tune and are training fast. Outstanding results.

Neural networks could likely outperform them, but require a lot of time and computing power.

Appendix

The following slides are appendix.

All code can be found here: https://github.com/bjarnemu/aml2021finalproject

General Notes

The models were trained on different systems, therefore the training times are not included, as it would not be directly comparable.

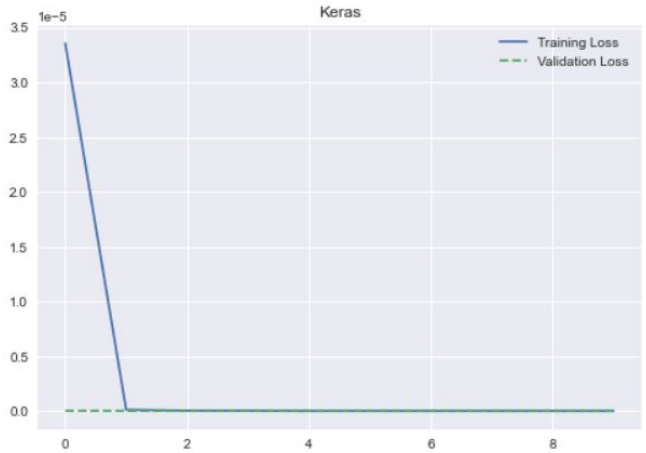
All predictions were saved using pickle, so the training process could be distributed on multiple systems.

Keras Sequential Neural Network

We tried a to make a sequential neural network.

Focal loss is a good loss function for imbalanced datasets[1]

However, results were not too promising



1 - Bhattacharyya, Saptashwa. “A Loss Function Suitable for Class Imbalanced Data: “Focal Loss” *Credit Medium* , 2021, https://towardsdatascience.com/a-loss-function-suitable-for-class-imbalanced-data-focal-loss-af1702d75d75

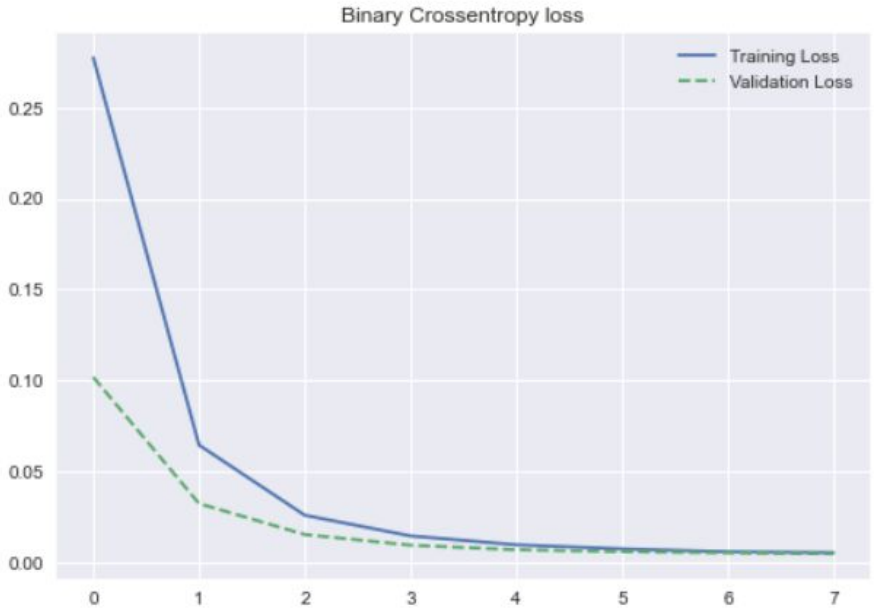
Keras Sequential Neural Network

We tuned the network!

Tried binary crossentropy as loss, Dropout layers, batch normalization, more layers, more neurons! Still bad results.

The issue was the validation data. It did not have enough positive cases.

Using half the training data as validation gave positive results!



Keras Neural Network

Was found to be quite hard to tune. All tuning was done manually using the official documentation as guideline.

Learning rate = 0.0001

Optimizer = Adam

Two dropout layers (rate = 0.4)

Focal Loss was scrapped due to SMOTE enabling usage of crossentropy.

Output neuron used sigmoid activation function, hidden layers used relu

Used early stopping after 5 epochs with validation loss not falling

Keras Neural Network

On average over all cross-folds, trained for…

- 24.8 epochs on all features
- 25.6 epochs on median difference features
- 26.6 epochs on permutation importance

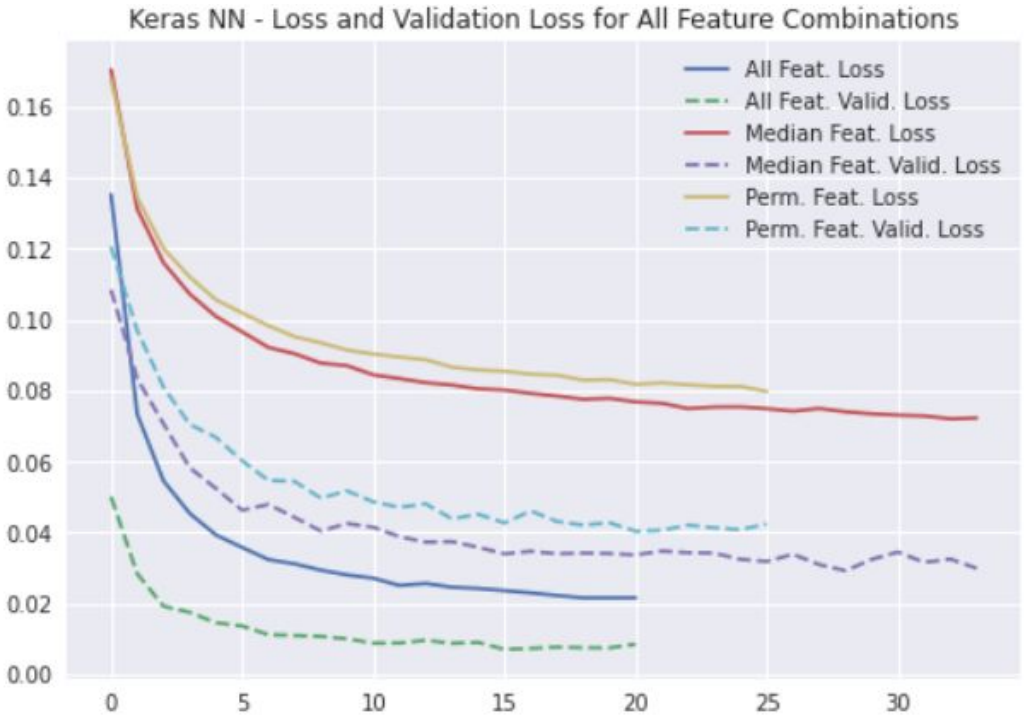
For speeding up training times, this model was trained on Kaggle’s servers

Keras Neural Network

Plotted training loss and validation loss for all feature combinations.

The loss on all features is dropping more rapidly and reaches a lower point.

The NN also has highest score on all features.

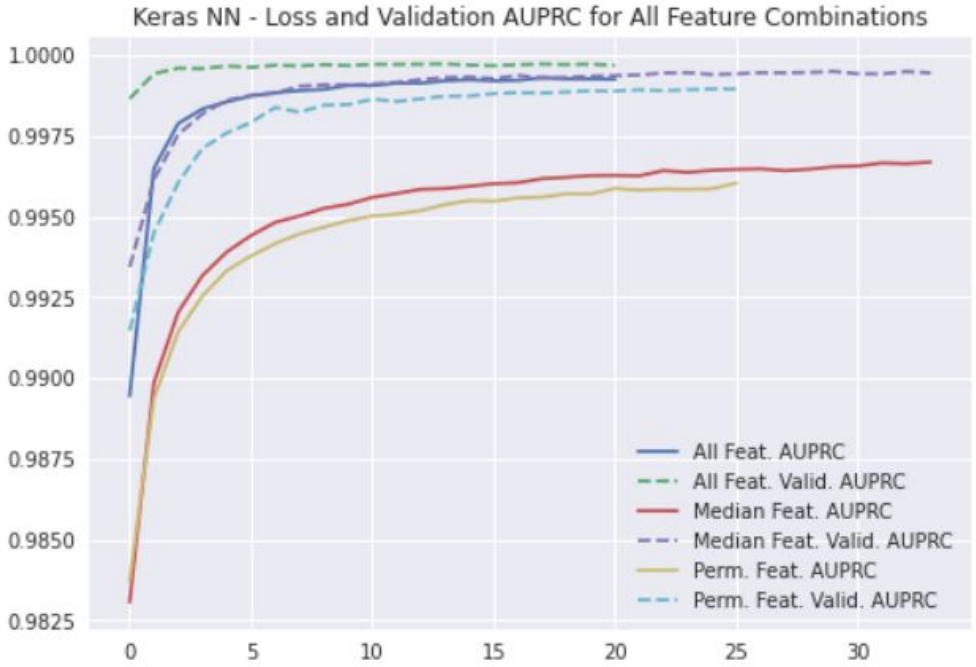


Keras Neural Network

Same a previous slide but showing AUPRC score during training.

Consistently with the losses, all feats has the highest score.

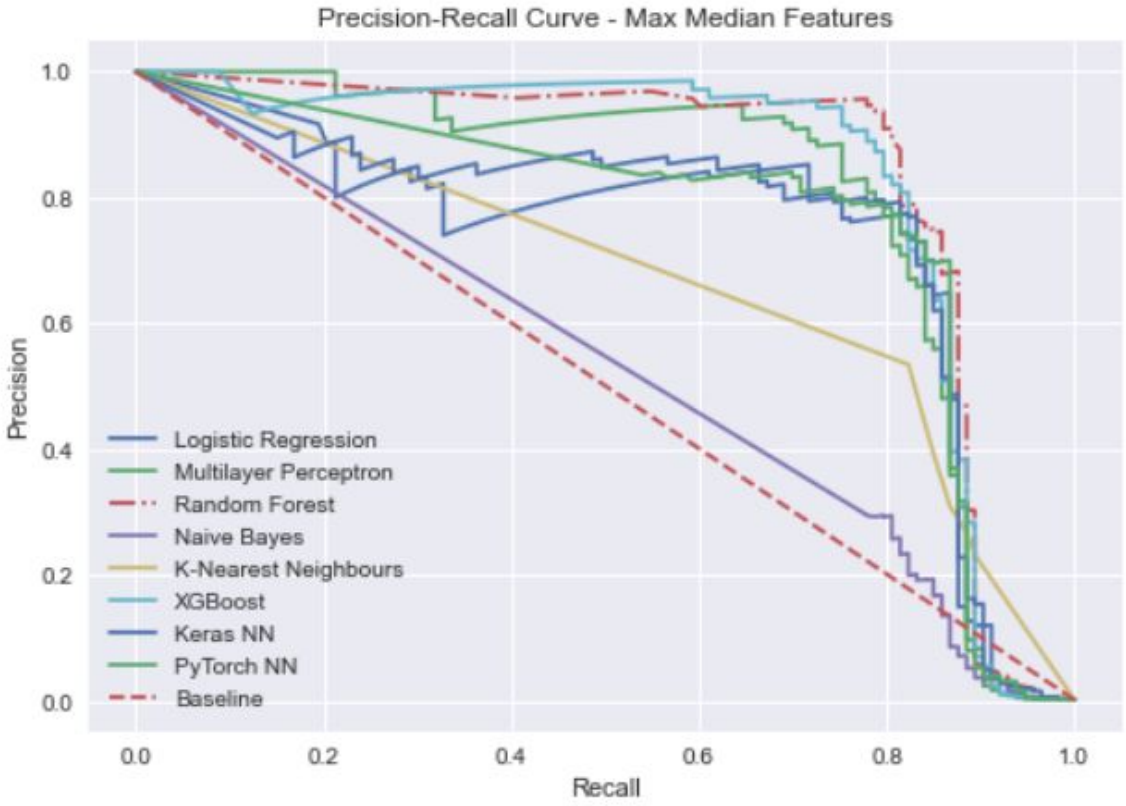
It quickly reached AUPRC scores close to 1.0 on the validation data.



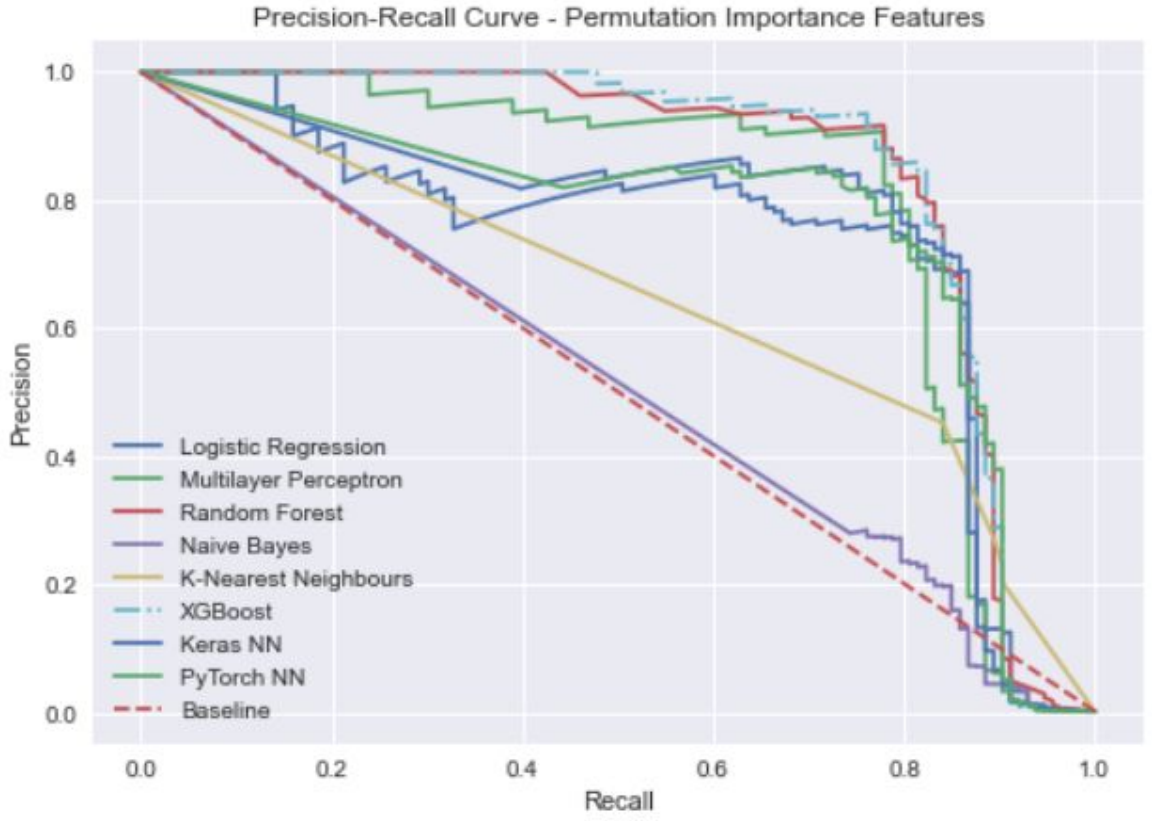
Additional Figures

The following slides are figures which were created, but not included in the presentation.

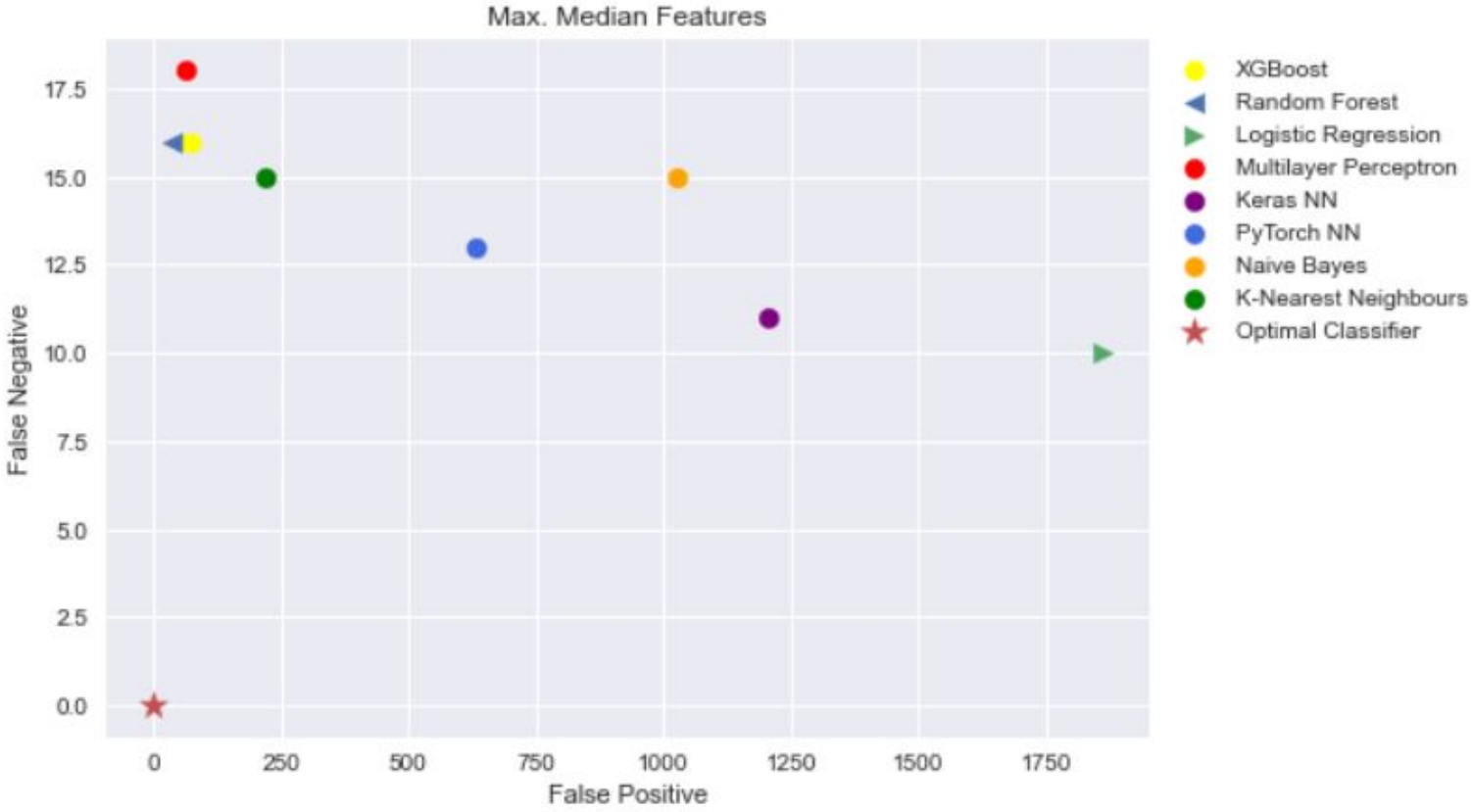
Precision-Recall Curve



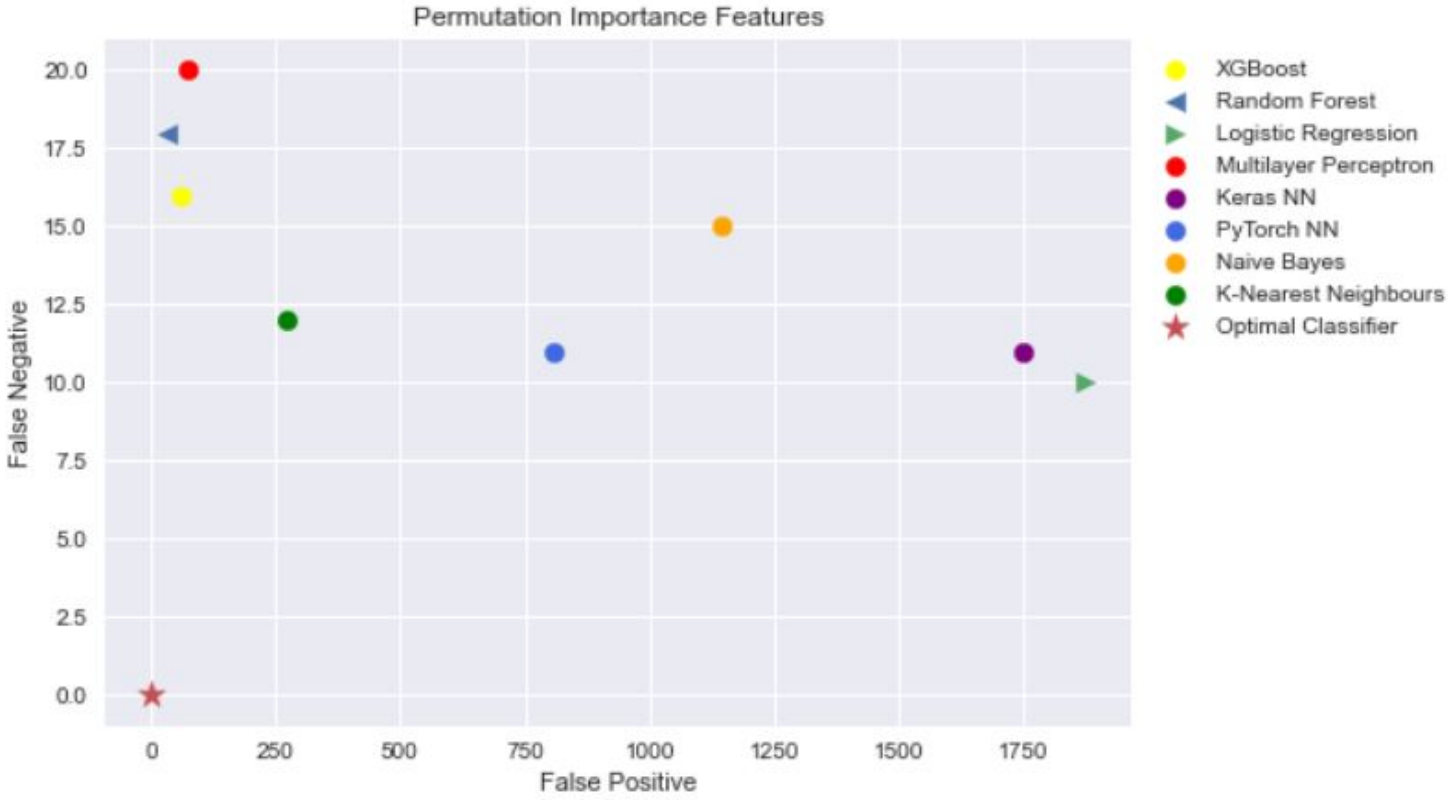
Precision-Recall Curve



False Positives and Negatives



False Positives and Negatives



Feature Combinations

We made three combinations of features, which each model was trained on.

1. Top 10 largest median difference
2. Top 10 permutation importance
3. All 28 features