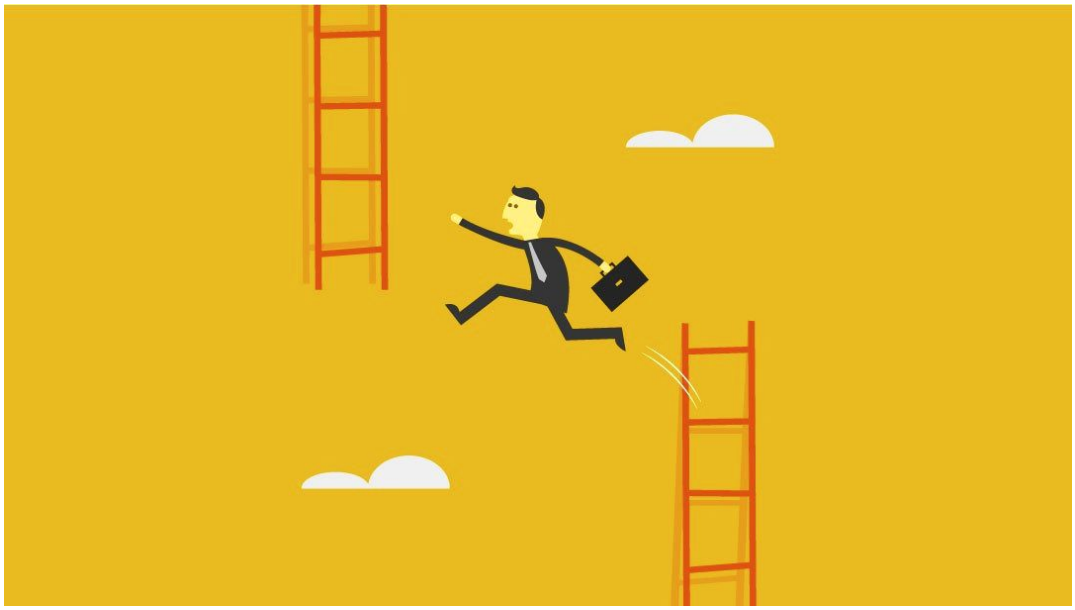
Layers and Livelihoods

Using Machine Learning to predict which Data Scientists are at risk of changing jobs

Team 1:
Max Eagle, Amy Ho, Ryan S. Wong

Hello everyone! We are Team One, comprised of Max Eagle, Amy Ho, and myself, Ryan Wong. And we're going to jump right into the most important lesson we learned from this project.



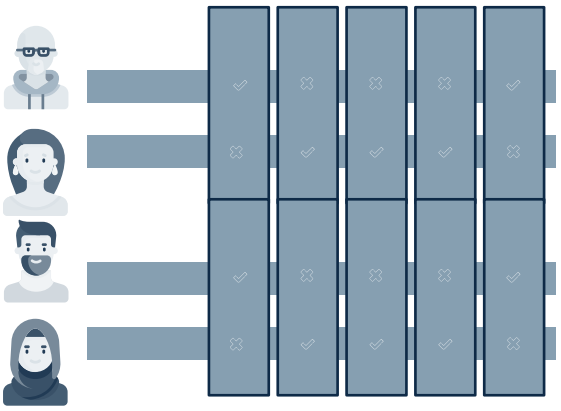
We learned that… The old ways of work are dead. Company loyalty is certain doom. The corporate ladder is broken. And if you want to move up, you have to get out first. That is the mantra of Silicon Valley tech work, a world moving so fast that staying in one place is falling behind. Following this mantra, individuals with highly demanded skills like Data Science are incentivized to change jobs constantly, thereby seeking greater pay, new skills, greater responsibilities, and more chances to advance their careers. To an eager, adaptable, mercenary-minded data scientist, this is the dream.

The Problem

High Employee Turnover
Operational Disruption
Financial Costs

But for the Human Resources departments interested in retaining those data scientists, this is a nightmare. High levels of employee turnover leads to staffing shortages, operational disruptions, and financial costs. As such, having a tool that would be able to predict what employees may be at risk of changing jobs can allow HR departments to be proactive and thus reduce employee turnover, operational disruption, and financial costs. And, in an ironic twist, data science provides such a tool for predicting which employees are at risk of changing jobs.

The Data



A survey of data scientists

We were first introduced to this problem after stumbling upon our data set. As burgeoning data scientists ourselves, it was the name that caught our eye: “Job Change of Data Scientists”.

“HR Analytics: Job Change of Data Scientists”

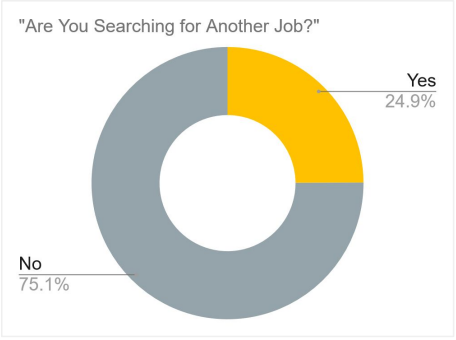
Data set from Kaggle

~21k Entries

(19k in Train, 2k in Test)

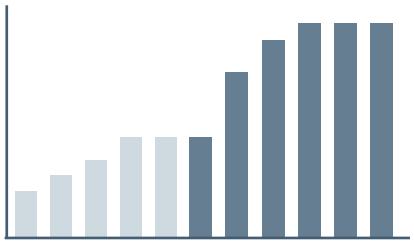
13 Features

2 numeric, 1 boolean, 9 categorical, 1 label



It was an open data set on Kaggle that was a derived from a larger HR Analytics data set. Survey responses from over 21 thousand data scientists across the United States were compiled together with 13 features of interest. And of those respondents, a baseline of about 25% of them indicated that they were indeed searching for another job.

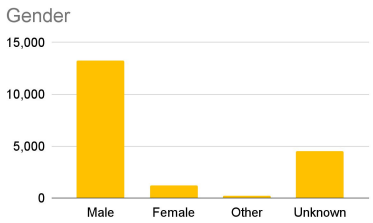
The EDA

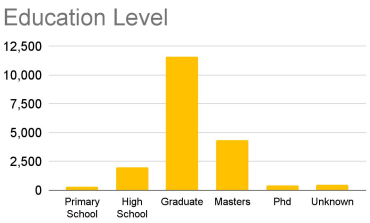


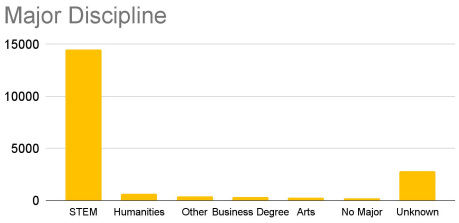
Imbalanced
Messy
Missing

We were really excited to get our feet wet with the data and start building models right away. But after we did our Exploratory Data Analysis, *TRIGGER* we learned that this data set was a lot like my dog: imbalanced, messy, and frequently missing.

Imbalanced







Imbalanced, because nearly every one of our 13 features- including Gender, Education Level, and Major Discipline as shown here- had imbalanced distributions that we would have to contend with.

Messy

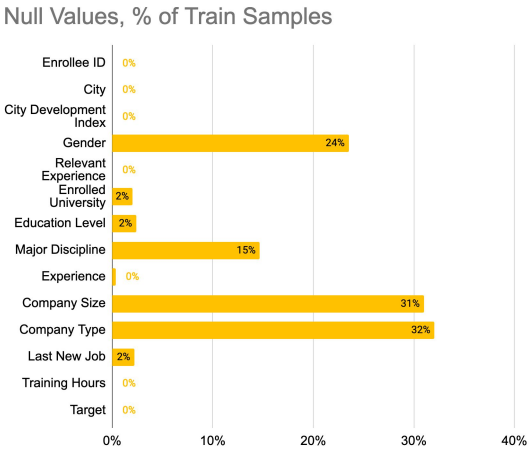
Too many categories!

Too many cities!

Typos. Typos Everywhere.

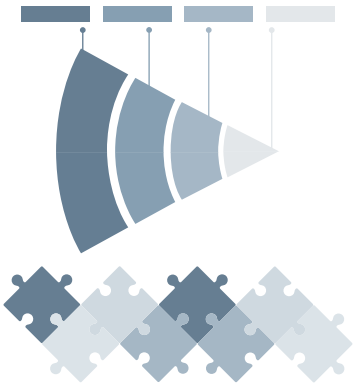
Unknowns,
Unknowns,
Unknowns.

Missing



Messy, because we found many oddities, ambiguities and straight-up typos in the data. And Missing, because some features had up to 32% of entries listed as null values.

The Approach



"It is estimated that data scientists spend about 80% of their time cleaning data." 😅

With so much cleaning necessary, we really identified with the saying that data scientists spend the majority of their time cleaning data. But with a little elbow grease and the right approach, we transformed our data in meaningful ways.

Rebalance, Clean, Encode

Rebalanced data with SMOTE
Equal split between label values

Applied numerical changes
"> 20 years" → 25 years
"< 1 year" → 0 years

One Hot Encoding on our features
Dropped the majority category

One-Hot Encoding Unknowns

We began by addressing the issues mentioned previously.

Imbalanced data was rebalanced with Synthetic Minority Oversampling Technique - AKA SMOTE. This works by randomly selecting an example from the minority class, finding the k nearest neighbors, and generating a synthetic example from these data points. After repeatedly doing this, we have perfectly balanced labels.

One example of applying numerical changes was a changing a categorical value of more than 20 years of experience into a value of 25 and adding a one-hot encoding that signified this as our overflow bin. We did the same for those with less than 1 year of experience.

Finally, we one-hot encoded all categorical features, dropped the majority category, and included unknowns to measure their impact.

Feature Selection

Dropped gender, major discipline, and company type
Skewed to one category with not a lot of variance

Principal Component Analysis (PCA)
Likely useful for decision tree and KNN

As we started working on the models, we also made two additional changes for feature selection.

We chose to drop gender, major discipline, and company type because they were not very influential in terms of feature importance.

We also implemented PCA for a second set of our models in case the reduced dimensionality might improve performance by decreasing the number of features and distance between data points. We suspected that this would be useful for Decision Trees or K Nearest Neighbors.

What Metrics to Use?

F1 + recall

+ Higher than 75% accuracy

Goal is to predict all positive cases.

False positives preferable to false negatives.

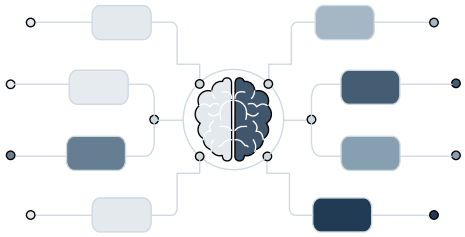
When evaluating our models, we decided that the F1 and recall scores were the most important ones to measure, but we did require that the models have an accuracy at least as good as the baseline. These metrics would allow us to maximize predictions for all positive cases while being cautious of accepting too many false positives.

From an business perspective, it's okay to predict that some people will leave even if they don't. But it's not okay to predict a person won't leave when they actually do.

What Model to Use?

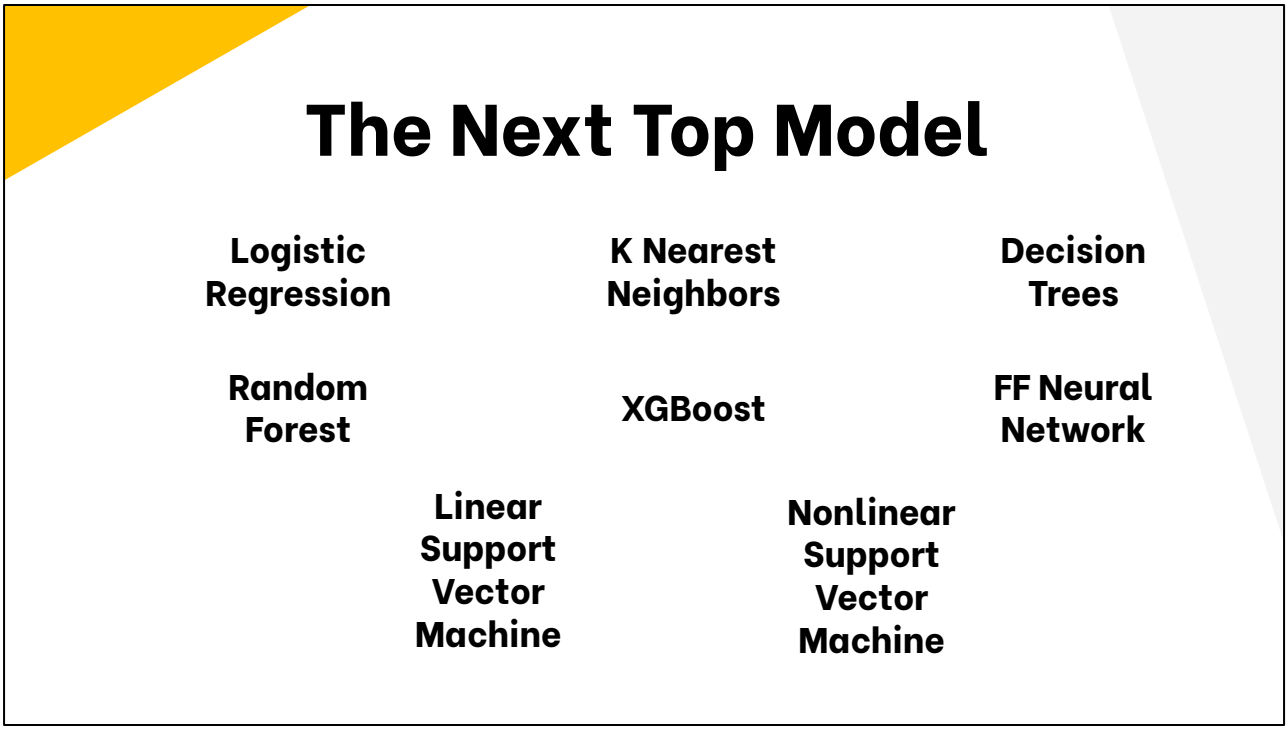
So with all of those decisions made, we just had to pick a model and run it, right? Well, truthfully, we weren't really sure which models would generate the best results. So we made a very sane, sensible decision: we were going to run every single model with every single combination of parameters and see what sticks. It took our three computers about 60 combined hours of burning silicon to actually pull off.

The Experiments



Models Hitting the Runway

With that plan settled, we let that spaghetti fly and looked at what stuck to the wall.



This is a list of the eight models that we chose to run. The top six you'll recognize from what we covered in the course, but the bottom two were recommended by John.

Wide Streets

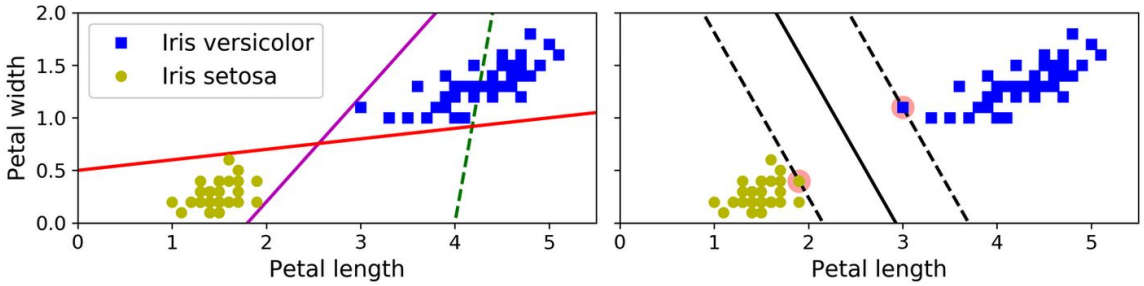
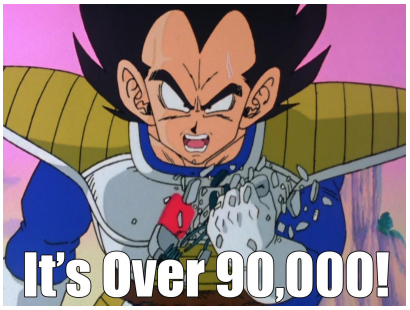


Figure 5-1. Large margin classification

In our supplemental research, we discovered that support vector machines work well because the decision boundaries they draw not only separate classes but also stay as far away from the closest training instances as possible. The idea here is to have large margins since this helps ensure better generalization on new data points.

Grid Search

Model Type	Hyperparameters
Decision Tree	criterion, max_depth, max_features
Feed Forward Neural Network	hidden_layers, neurons, activation, optimizer, learning_rate, epochs, batch_size
K Nearest Neighbors	n_neighbors, weights, algorithm, leaf_size, p
Linear Support Vector Machine	penalty, C
Logistic Regression	penalty, solver, C, l1_ratio
Non-linear Support Vector Machine	C, degree, coef0
Random Forest	criterion, max_depth, max_features, n_estimators
XGBoost	learning_rate, max_depth, n_estimators

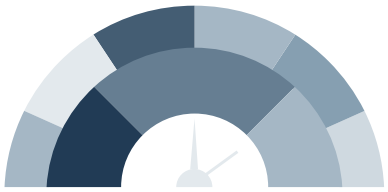


Each of those eight models were further tuned with hyperparameter combinations via Grid Search. Listed here are all of the models and the hyperparameters that we applied. We collected and measured the results of each model using the initial parameters, initial parameters with PCA, and with sets of parameters that maximize Area Under the Curve, F1, and recall scores.

Fun fact: because of the number of combinations possible, running all 8 models with the full suite of grid searches resulted in 102,836 models being created and fit. That's a lot of data! Hence Vegeta's shock and crushing his scouter.

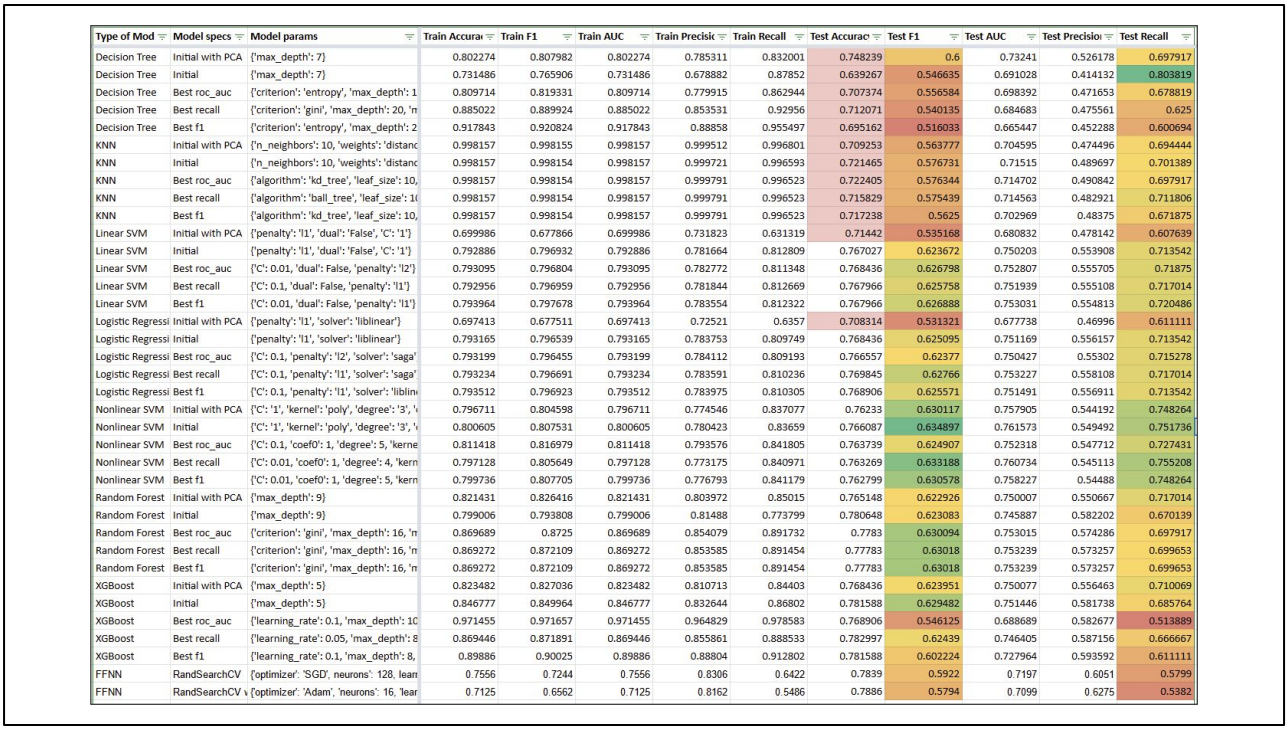
Number of Parameters Math:
DT: 3*4*2 = 12
KNN: 9*2*4*2*2 = 288
FFNN: 4*10*2*2*3*10*8 = 38400
LR: 4*14*35*28 = 49000
RF: 3*4*2*4 = 96
Xgboost: 5*5*4 = 100
LSVM: 2*1*7 = 14
SVM: 5*4*2 = 40
TOTAL: 102836

The Results



Judging Begins...

Running all those models took a LOT of time, mostly on Ryan and Max's computers because my computer is the least chonky of all of ours. After all that machine runtime, we had our results.



Type of Mod	Model specs	Model params	Train Accura	Train F1	Train AUC	Train Precisi	Train Recall	Test Accurac	Test F1	Test AUC	Test Precisio	Test Recall
Decision Tree	Initial with PCA	{'max_depth': 7}	0.802274	0.807982	0.802274	0.785311	0.832001	0.748239	0.6	0.73241	0.526178	0.697917
Decision Tree	Initial	{'max_depth': 7}	0.731486	0.765906	0.731486	0.678882	0.87852	0.639267	0.546635	0.691028	0.414132	0.803819
Decision Tree	Best roc_auc	{'criterion': 'entropy', 'max_depth': 1	0.809714	0.819331	0.809714	0.779915	0.862944	0.707374	0.556584	0.698392	0.471653	0.678819
Decision Tree	Best recall	{'criterion': 'gini', 'max_depth': 20, 'n	0.885022	0.889924	0.885022	0.853531	0.92956	0.712071	0.540135	0.684683	0.475561	0.625
Decision Tree	Best f1	{'criterion': 'entropy', 'max_depth': 2	0.917843	0.920824	0.917843	0.88858	0.955497	0.695162	0.516033	0.665447	0.452288	0.600694
KNN	Initial with PCA	{'n_neighbors': 10, 'weights': 'distanc	0.998157	0.998155	0.998157	0.999512	0.996801	0.709253	0.563777	0.704595	0.474496	0.694444
KNN	Initial	{'n_neighbors': 10, 'weights': 'distanc	0.998157	0.998154	0.998157	0.999721	0.996593	0.721465	0.576731	0.71515	0.489697	0.701389
KNN	Best roc_auc	{'algorithm': 'kd_tree', 'leaf_size': 10,	0.998157	0.998154	0.998157	0.999791	0.996523	0.722405	0.576344	0.714702	0.490842	0.697917
KNN	Best recall	{'algorithm': 'ball_tree', 'leaf_size': 1(	0.998157	0.998154	0.998157	0.999791	0.996523	0.715829	0.575439	0.714563	0.482921	0.711806
KNN	Best f1	{'algorithm': 'kd_tree', 'leaf_size': 10,	0.998157	0.998154	0.998157	0.999791	0.996523	0.717238	0.5625	0.702969	0.48375	0.671875
Linear SVM	Initial with PCA	{'penalty': 'l1', 'dual': 'False', 'C': '1'}	0.699986	0.677866	0.699986	0.731823	0.631319	0.71442	0.535168	0.680832	0.478142	0.607639
Linear SVM	Initial	{'penalty': 'l1', 'dual': 'False', 'C': '1'}	0.792886	0.796932	0.792886	0.781664	0.812809	0.767027	0.623672	0.750203	0.553908	0.713542
Linear SVM	Best roc_auc	{'C': 0.01, 'dual': False, 'penalty': 'l2'}	0.793095	0.796804	0.793095	0.782772	0.811348	0.768436	0.626798	0.752807	0.555705	0.71875
Linear SVM	Best recall	{'C': 0.1, 'dual': False, 'penalty': 'l1'}	0.792956	0.796959	0.792956	0.781844	0.812669	0.767966	0.625758	0.751939	0.555108	0.717014
Linear SVM	Best f1	{'C': 0.01, 'dual': False, 'penalty': 'l1'}	0.793964	0.797678	0.793964	0.783554	0.812322	0.767966	0.626888	0.753031	0.554813	0.720486
Logistic Regressi	Initial with PCA	{'penalty': 'l1', 'solver': 'liblinear'}	0.697413	0.677511	0.697413	0.72521	0.6357	0.708314	0.531321	0.677738	0.46996	0.611111
Logistic Regressi	Initial	{'penalty': 'l1', 'solver': 'liblinear'}	0.793165	0.796539	0.793165	0.783753	0.809749	0.768436	0.625095	0.751169	0.556157	0.713542
Logistic Regressi	Best roc_auc	{'C': 0.1, 'penalty': 'l2', 'solver': 'saga'	0.793199	0.796455	0.793199	0.784112	0.809193	0.766557	0.62377	0.750427	0.55302	0.715278
Logistic Regressi	Best recall	{'C': 0.1, 'penalty': 'l1', 'solver': 'saga'	0.793234	0.796691	0.793234	0.783591	0.810236	0.769845	0.62766	0.753227	0.558108	0.717014
Logistic Regressi	Best f1	{'C': 0.1, 'penalty': 'l1', 'solver': 'liblin	0.793512	0.796923	0.793512	0.783975	0.810305	0.768906	0.625571	0.751491	0.556911	0.713542
Nonlinear SVM	Initial with PCA	{'C': '1', 'kernel': 'poly', 'degree': '3', '	0.796711	0.804598	0.796711	0.774546	0.837077	0.76233	0.630117	0.757905	0.544192	0.748264
Nonlinear SVM	Initial	{'C': '1', 'kernel': 'poly', 'degree': '3', '	0.800605	0.807531	0.800605	0.780423	0.83659	0.766087	0.634897	0.761573	0.549492	0.751736
Nonlinear SVM	Best roc_auc	{'C': 0.1, 'coef0': 1, 'degree': 5, 'kerne	0.811418	0.816979	0.811418	0.793576	0.841805	0.763739	0.624907	0.752318	0.547712	0.727431
Nonlinear SVM	Best recall	{'C': 0.01, 'coef0': 1, 'degree': 4, 'kern	0.797128	0.805649	0.797128	0.773175	0.840971	0.763269	0.633188	0.760734	0.545113	0.755208
Nonlinear SVM	Best f1	{'C': 0.01, 'coef0': 1, 'degree': 5, 'kern	0.799736	0.807705	0.799736	0.776793	0.841179	0.762799	0.630578	0.758227	0.54488	0.748264
Random Forest	Initial with PCA	{'max_depth': 9}	0.821431	0.826416	0.821431	0.803972	0.85015	0.765148	0.622926	0.750007	0.550667	0.717014
Random Forest	Initial	{'max_depth': 9}	0.799006	0.793808	0.799006	0.81488	0.773799	0.780648	0.623083	0.745887	0.582202	0.670139
Random Forest	Best roc_auc	{'criterion': 'gini', 'max_depth': 16, 'n	0.869689	0.8725	0.869689	0.854079	0.891732	0.7783	0.630094	0.753015	0.574286	0.697917
Random Forest	Best recall	{'criterion': 'gini', 'max_depth': 16, 'n	0.869272	0.872109	0.869272	0.853585	0.891454	0.77783	0.63018	0.753239	0.573257	0.699653
Random Forest	Best f1	{'criterion': 'gini', 'max_depth': 16, 'n	0.869272	0.872109	0.869272	0.853585	0.891454	0.77783	0.63018	0.753239	0.573257	0.699653
XGBoost	Initial with PCA	{'max_depth': 5}	0.823482	0.827036	0.823482	0.810713	0.84403	0.768436	0.623951	0.750077	0.556463	0.710069
XGBoost	Initial	{'max_depth': 5}	0.846777	0.849964	0.846777	0.832644	0.86802	0.781588	0.629482	0.751446	0.581738	0.685764
XGBoost	Best roc_auc	{'learning_rate': 0.1, 'max_depth': 10	0.971455	0.971657	0.971455	0.964829	0.978583	0.768906	0.546125	0.688689	0.582677	0.513889
XGBoost	Best recall	{'learning_rate': 0.05, 'max_depth': 8	0.869446	0.871891	0.869446	0.855861	0.888533	0.782997	0.62439	0.746405	0.587156	0.666667
XGBoost	Best f1	{'learning_rate': 0.1, 'max_depth': 8,	0.89886	0.90025	0.89886	0.88804	0.912802	0.781588	0.602224	0.727964	0.593592	0.611111
FFNN	RandSearchCV	{'optimizer': 'SGD', 'neurons': 128, 'lear	0.7556	0.7244	0.7556	0.8306	0.6422	0.7839	0.5922	0.7197	0.6051	0.5799
FFNN	RandSearchCV v	{'optimizer': 'Adam', 'neurons': 16, 'lear	0.7125	0.6562	0.7125	0.8162	0.5486	0.7886	0.5794	0.7099	0.6275	0.5382

This is one tab of our results scoring spreadsheet, which collected the scores from each of our models. The runs were non-deterministic, so the scores varied a little each time. We mostly cared about F1 and recall scores, which are the color-coded boxes, with green being higher scores and red being lower scores. The test accuracy column just flags which models returned a worse accuracy than our baseline of 75%.

You'll note that the best test recall score was also paired with pretty poor F1 and accuracy scores, so we didn't just want to pick a model with the best singular score. We eliminated models that didn't beat our baseline accuracy, and then balanced F1 and recall.

Model Evaluation

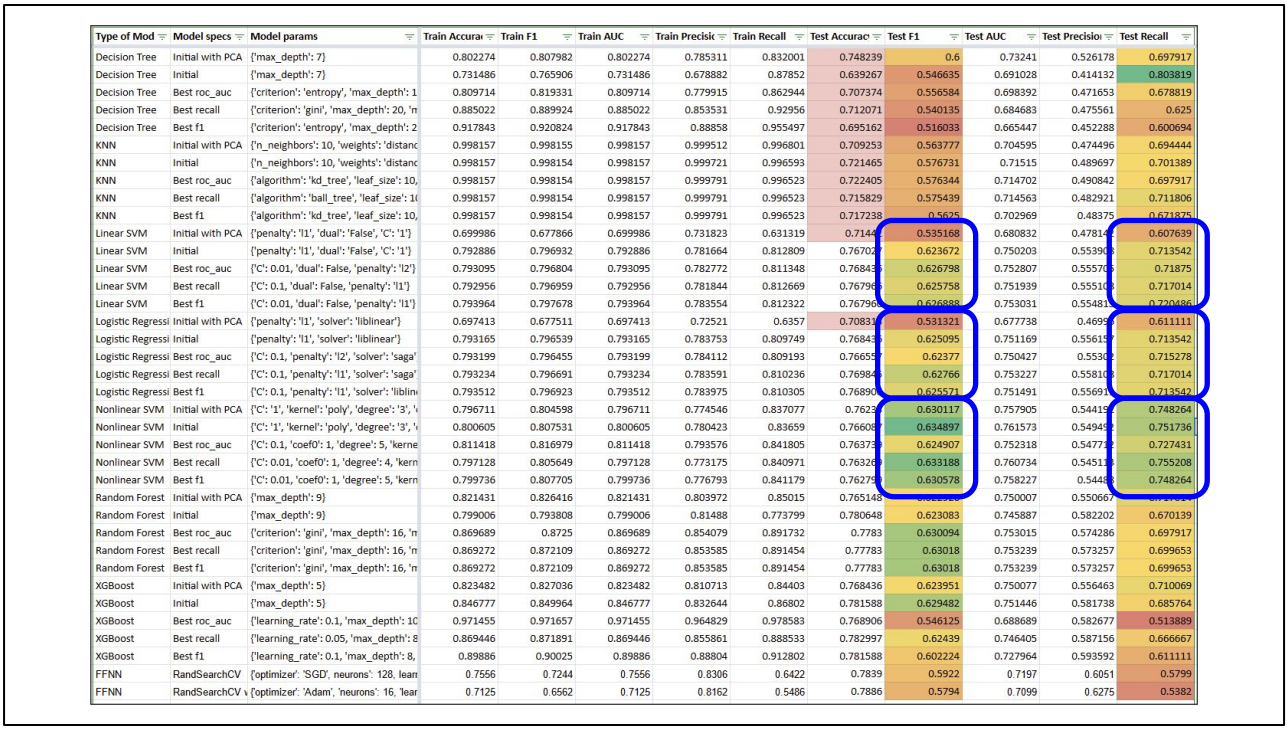
Best models :

Nonlinear SVM

Runner-up is split between Logistic Regression and Linear SVM

This led us to the nonlinear support vector machine models. After we eliminated any models that didn't beat our baseline accuracy score, the nonlinear SVM models had both the best F1 and recall scores. They only had a slightly higher, like 1.5% higher, accuracy than our baseline model.

The logistic regression and linear SVM models also did well, but were just slightly bested by the nonlinear SVM.



Type of Mod	Model specs	Model params	Train Accura	Train F1	Train AUC	Train Precisi	Train Recall	Test Accurac	Test F1	Test AUC	Test Precisio	Test Recall
Decision Tree	Initial with PCA	{'max_depth': 7}	0.802274	0.807982	0.802274	0.785311	0.832001	0.748239	0.6	0.73241	0.526178	0.697917
Decision Tree	Initial	{'max_depth': 7}	0.731486	0.765906	0.731486	0.678882	0.87852	0.639267	0.546635	0.691028	0.414132	0.803819
Decision Tree	Best roc_auc	{'criterion': 'entropy', 'max_depth': 1	0.809714	0.819331	0.809714	0.779915	0.862944	0.707374	0.556584	0.698392	0.471653	0.678819
Decision Tree	Best recall	{'criterion': 'gini', 'max_depth': 20, 'n	0.885022	0.889924	0.885022	0.853531	0.92956	0.712071	0.540135	0.684683	0.475561	0.625
Decision Tree	Best f1	{'criterion': 'entropy', 'max_depth': 2	0.917843	0.920824	0.917843	0.88858	0.955497	0.695162	0.516033	0.665447	0.452288	0.600694
KNN	Initial with PCA	{'n_neighbors': 10, 'weights': 'distanc	0.998157	0.998155	0.998157	0.999512	0.996801	0.709253	0.563777	0.704595	0.474496	0.694444
KNN	Initial	{'n_neighbors': 10, 'weights': 'distanc	0.998157	0.998154	0.998157	0.999721	0.996593	0.721465	0.576731	0.71515	0.489697	0.701389
KNN	Best roc_auc	{'algorithm': 'kd_tree', 'leaf_size': 10,	0.998157	0.998154	0.998157	0.999791	0.996523	0.722405	0.576344	0.714702	0.490842	0.697917
KNN	Best recall	{'algorithm': 'ball_tree', 'leaf_size': 1(	0.998157	0.998154	0.998157	0.999791	0.996523	0.715829	0.575439	0.714563	0.482921	0.711806
KNN	Best f1	{'algorithm': 'kd_tree', 'leaf_size': 10,	0.998157	0.998154	0.998157	0.999791	0.996523	0.717238	0.5625	0.702969	0.48375	0.671875
Linear SVM	Initial with PCA	{'penalty': 'l1', 'dual': 'False', 'C': '1'}	0.699986	0.677866	0.699986	0.731823	0.631319	0.7144	0.535168	0.680832	0.47814	0.607639
Linear SVM	Initial	{'penalty': 'l1', 'dual': 'False', 'C': '1'}	0.792886	0.796932	0.792886	0.781664	0.812809	0.76702	0.623672	0.750203	0.55390	0.713542
Linear SVM	Best roc_auc	{'C': 0.01, 'dual': False, 'penalty': 'l2'}	0.793095	0.796804	0.793095	0.782772	0.811348	0.76843	0.626798	0.752807	0.55570	0.71875
Linear SVM	Best recall	{'C': 0.1, 'dual': False, 'penalty': 'l1'}	0.792956	0.796959	0.792956	0.781844	0.812669	0.76796	0.625758	0.751939	0.55510	0.717014
Linear SVM	Best f1	{'C': 0.01, 'dual': False, 'penalty': 'l1'}	0.793964	0.797678	0.793964	0.783554	0.812322	0.76796	0.626888	0.753031	0.55481	0.720486
Logistic Regressi	Initial with PCA	{'penalty': 'l1', 'solver': 'liblinear'}	0.697413	0.677511	0.697413	0.72521	0.6357	0.70831	0.531321	0.677738	0.4699	0.611111
Logistic Regressi	Initial	{'penalty': 'l1', 'solver': 'liblinear'}	0.793165	0.796539	0.793165	0.783753	0.809749	0.76843	0.625095	0.751169	0.55615	0.713542
Logistic Regressi	Best roc_auc	{'C': 0.1, 'penalty': 'l2', 'solver': 'saga'	0.793199	0.796455	0.793199	0.784112	0.809193	0.76655	0.62377	0.750427	0.5530	0.715278
Logistic Regressi	Best recall	{'C': 0.1, 'penalty': 'l1', 'solver': 'saga'	0.793234	0.796691	0.793234	0.783591	0.810236	0.76984	0.62766	0.753227	0.55810	0.717014
Logistic Regressi	Best f1	{'C': 0.1, 'penalty': 'l1', 'solver': 'liblin	0.793512	0.796923	0.793512	0.783975	0.810305	0.76890	0.625571	0.751491	0.55691	0.713542
Nonlinear SVM	Initial with PCA	{'C': '1', 'kernel': 'poly', 'degree': '3', '	0.796711	0.804598	0.796711	0.774546	0.837077	0.7623	0.630117	0.757905	0.54419	0.748264
Nonlinear SVM	Initial	{'C': '1', 'kernel': 'poly', 'degree': '3', '	0.800605	0.807531	0.800605	0.780423	0.83659	0.76608	0.634897	0.761573	0.54949	0.751736
Nonlinear SVM	Best roc_auc	{'C': 0.1, 'coef0': 1, 'degree': 5, 'kerne	0.811418	0.816979	0.811418	0.793576	0.841805	0.76373	0.624907	0.752318	0.54773	0.727431
Nonlinear SVM	Best recall	{'C': 0.01, 'coef0': 1, 'degree': 4, 'kern	0.797128	0.805649	0.797128	0.773175	0.840971	0.76326	0.633188	0.760734	0.54511	0.755208
Nonlinear SVM	Best f1	{'C': 0.01, 'coef0': 1, 'degree': 5, 'kern	0.799736	0.807705	0.799736	0.776793	0.841179	0.76275	0.630578	0.758227	0.5448	0.748264
Random Forest	Initial with PCA	{'max_depth': 9}	0.821431	0.826416	0.821431	0.803972	0.85015	0.765148		0.750007	0.550667	
Random Forest	Initial	{'max_depth': 9}	0.799006	0.793808	0.799006	0.81488	0.773799	0.780648	0.623083	0.745887	0.582202	0.670139
Random Forest	Best roc_auc	{'criterion': 'gini', 'max_depth': 16, 'n	0.869689	0.8725	0.869689	0.854079	0.891732	0.7783	0.630094	0.753015	0.574286	0.697917
Random Forest	Best recall	{'criterion': 'gini', 'max_depth': 16, 'n	0.869272	0.872109	0.869272	0.853585	0.891454	0.77783	0.63018	0.753239	0.573257	0.699653
Random Forest	Best f1	{'criterion': 'gini', 'max_depth': 16, 'n	0.869272	0.872109	0.869272	0.853585	0.891454	0.77783	0.63018	0.753239	0.573257	0.699653
XGBoost	Initial with PCA	{'max_depth': 5}	0.823482	0.827036	0.823482	0.810713	0.84403	0.768436	0.623951	0.750077	0.556463	0.710069
XGBoost	Initial	{'max_depth': 5}	0.846777	0.849964	0.846777	0.832644	0.86802	0.781588	0.629482	0.751446	0.581738	0.685764
XGBoost	Best roc_auc	{'learning_rate': 0.1, 'max_depth': 10	0.971455	0.971657	0.971455	0.964829	0.978583	0.768906	0.546125	0.688689	0.582677	0.513889
XGBoost	Best recall	{'learning_rate': 0.05, 'max_depth': 8	0.869446	0.871891	0.869446	0.855861	0.888533	0.782997	0.62439	0.746405	0.587156	0.666667
XGBoost	Best f1	{'learning_rate': 0.1, 'max_depth': 8,	0.89886	0.90025	0.89886	0.88804	0.912802	0.781588	0.602224	0.727964	0.593592	0.611111
FFNN	RandSearchCV	{'optimizer': 'SGD', 'neurons': 128, 'lear	0.7556	0.7244	0.7556	0.8306	0.6422	0.7839	0.5922	0.7197	0.6051	0.5799
FFNN	RandSearchCV v	{'optimizer': 'Adam', 'neurons': 16, 'lear	0.7125	0.6562	0.7125	0.8162	0.5486	0.7886	0.5794	0.7099	0.6275	0.5382

Here you can see that the nonlinear SVM models, which is the group of scores that's circled lowest on this page, had recall scores that hovered around 75%, compared with something more like 71-2% for logistic regression and linear SVM. And the F1 scores for nonlinear SVM are also higher, but there's not as big of a difference.

Model Evaluation

Best models :

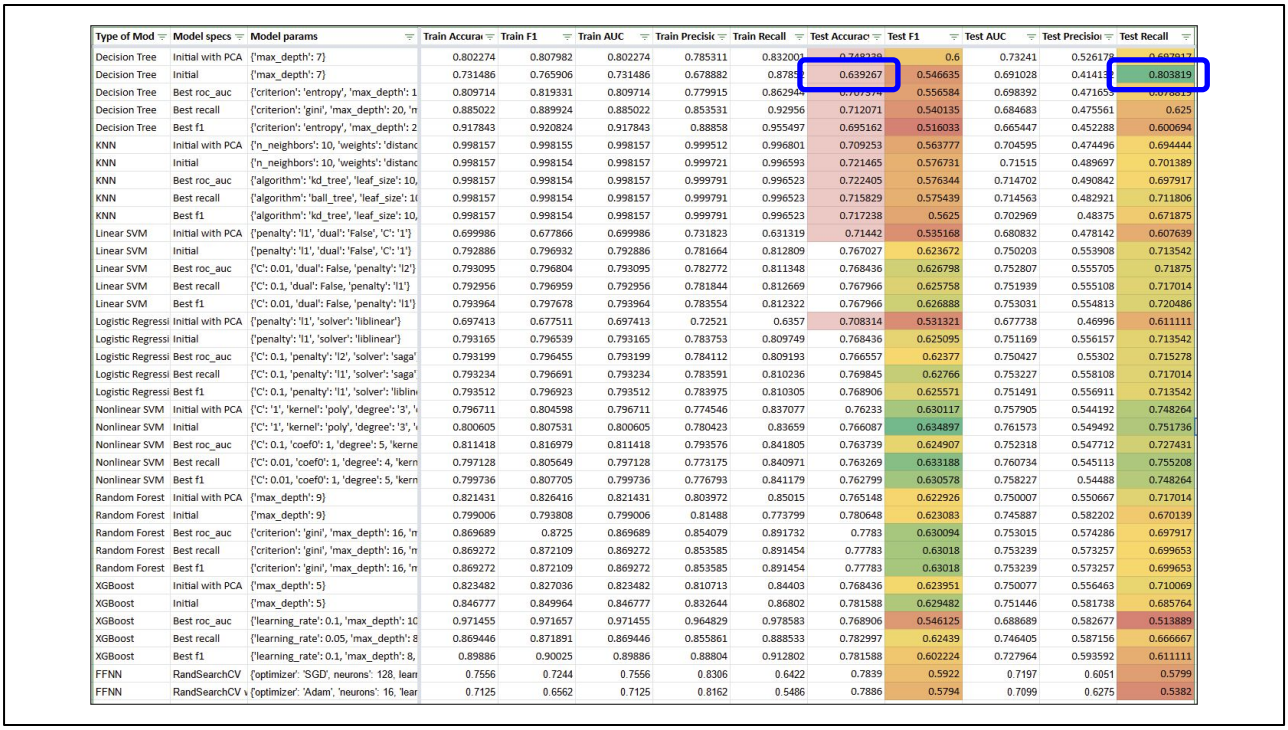
Nonlinear SVM

Runner-up is split between Logistic Regression and Linear SVM

Notes of interest:

Decision tree recall the best but awful accuracy

Beyond just what we decided were our best models for our purposes, we also noted that decision trees had the best recall score, of around 80%, but had one of the worst accuracies.



Type of Mod	Model specs	Model params	Train Accura	Train F1	Train AUC	Train Precisi	Train Recall	Test Accurac	Test F1	Test AUC	Test Precisio	Test Recall
Decision Tree	Initial with PCA	{'max_depth': 7}	0.802274	0.807982	0.802274	0.785311	0.832001	0.748338	0.6	0.73241	0.526178	0.697917
Decision Tree	Initial	{'max_depth': 7}	0.731486	0.765906	0.731486	0.678882	0.8785	0.639267	0.546635	0.691028	0.414133	0.803819
Decision Tree	Best roc_auc	{'criterion': 'entropy', 'max_depth': 1	0.809714	0.819331	0.809714	0.779915	0.862944	0.707374	0.556584	0.698392	0.471655	0.678819
Decision Tree	Best recall	{'criterion': 'gini', 'max_depth': 20, 'n	0.885022	0.889924	0.885022	0.853531	0.92956	0.712071	0.540135	0.684683	0.475561	0.625
Decision Tree	Best f1	{'criterion': 'entropy', 'max_depth': 2	0.917843	0.920824	0.917843	0.88858	0.955497	0.695162	0.516033	0.665447	0.452288	0.600694
KNN	Initial with PCA	{'n_neighbors': 10, 'weights': 'distanc	0.998157	0.998155	0.998157	0.999512	0.996801	0.709253	0.563777	0.704595	0.474496	0.694444
KNN	Initial	{'n_neighbors': 10, 'weights': 'distanc	0.998157	0.998154	0.998157	0.999721	0.996593	0.721465	0.576731	0.71515	0.489697	0.701389
KNN	Best roc_auc	{'algorithm': 'kd_tree', 'leaf_size': 10,	0.998157	0.998154	0.998157	0.999791	0.996523	0.722405	0.576344	0.714702	0.490842	0.697917
KNN	Best recall	{'algorithm': 'ball_tree', 'leaf_size': 1(	0.998157	0.998154	0.998157	0.999791	0.996523	0.715829	0.575439	0.714563	0.482921	0.711806
KNN	Best f1	{'algorithm': 'kd_tree', 'leaf_size': 10,	0.998157	0.998154	0.998157	0.999791	0.996523	0.717238	0.5625	0.702969	0.48375	0.671875
Linear SVM	Initial with PCA	{'penalty': 'l1', 'dual': 'False', 'C': '1'}	0.699986	0.677866	0.699986	0.731823	0.631319	0.71442	0.535168	0.680832	0.478142	0.607639
Linear SVM	Initial	{'penalty': 'l1', 'dual': 'False', 'C': '1'}	0.792886	0.796932	0.792886	0.781664	0.812809	0.767027	0.623672	0.750203	0.553908	0.713542
Linear SVM	Best roc_auc	{'C': 0.01, 'dual': False, 'penalty': 'l2'}	0.793095	0.796804	0.793095	0.782772	0.811348	0.768436	0.626798	0.752807	0.555705	0.71875
Linear SVM	Best recall	{'C': 0.1, 'dual': False, 'penalty': 'l1'}	0.792956	0.796959	0.792956	0.781844	0.812669	0.767966	0.625758	0.751939	0.555108	0.717014
Linear SVM	Best f1	{'C': 0.01, 'dual': False, 'penalty': 'l1'}	0.793964	0.797678	0.793964	0.783554	0.812322	0.767966	0.626888	0.753031	0.554813	0.720486
Logistic Regressi	Initial with PCA	{'penalty': 'l1', 'solver': 'liblinear'}	0.697413	0.677511	0.697413	0.72521	0.6357	0.708314	0.531321	0.677738	0.46996	0.611111
Logistic Regressi	Initial	{'penalty': 'l1', 'solver': 'liblinear'}	0.793165	0.796539	0.793165	0.783753	0.809749	0.768436	0.625095	0.751169	0.556157	0.713542
Logistic Regressi	Best roc_auc	{'C': 0.1, 'penalty': 'l2', 'solver': 'saga'	0.793199	0.796455	0.793199	0.784112	0.809193	0.766557	0.62377	0.750427	0.55302	0.715278
Logistic Regressi	Best recall	{'C': 0.1, 'penalty': 'l1', 'solver': 'saga'	0.793234	0.796691	0.793234	0.783591	0.810236	0.769845	0.62766	0.753227	0.558108	0.717014
Logistic Regressi	Best f1	{'C': 0.1, 'penalty': 'l1', 'solver': 'liblin	0.793512	0.796923	0.793512	0.783975	0.810305	0.768906	0.625571	0.751491	0.556911	0.713542
Nonlinear SVM	Initial with PCA	{'C': '1', 'kernel': 'poly', 'degree': '3', '	0.796711	0.804598	0.796711	0.774546	0.837077	0.76233	0.630117	0.757905	0.544192	0.748264
Nonlinear SVM	Initial	{'C': '1', 'kernel': 'poly', 'degree': '3', '	0.800605	0.807531	0.800605	0.780423	0.83659	0.766087	0.634897	0.761573	0.549492	0.751736
Nonlinear SVM	Best roc_auc	{'C': 0.1, 'coef0': 1, 'degree': 5, 'kerne	0.811418	0.816979	0.811418	0.793576	0.841805	0.763739	0.624907	0.752318	0.547712	0.727431
Nonlinear SVM	Best recall	{'C': 0.01, 'coef0': 1, 'degree': 4, 'kern	0.797128	0.805649	0.797128	0.773175	0.840971	0.763269	0.633188	0.760734	0.545113	0.755208
Nonlinear SVM	Best f1	{'C': 0.01, 'coef0': 1, 'degree': 5, 'kern	0.799736	0.807705	0.799736	0.776793	0.841179	0.762799	0.630578	0.758227	0.54488	0.748264
Random Forest	Initial with PCA	{'max_depth': 9}	0.821431	0.826416	0.821431	0.803972	0.85015	0.765148	0.622926	0.750007	0.550667	0.717014
Random Forest	Initial	{'max_depth': 9}	0.799006	0.793808	0.799006	0.81488	0.773799	0.780648	0.623083	0.745887	0.582202	0.670139
Random Forest	Best roc_auc	{'criterion': 'gini', 'max_depth': 16, 'n	0.869689	0.8725	0.869689	0.854079	0.891732	0.7783	0.630094	0.753015	0.574286	0.697917
Random Forest	Best recall	{'criterion': 'gini', 'max_depth': 16, 'n	0.869272	0.872109	0.869272	0.853585	0.891454	0.77783	0.63018	0.753239	0.573257	0.699653
Random Forest	Best f1	{'criterion': 'gini', 'max_depth': 16, 'n	0.869272	0.872109	0.869272	0.853585	0.891454	0.77783	0.63018	0.753239	0.573257	0.699653
XGBoost	Initial with PCA	{'max_depth': 5}	0.823482	0.827036	0.823482	0.810713	0.84403	0.768436	0.623951	0.750077	0.556463	0.710069
XGBoost	Initial	{'max_depth': 5}	0.846777	0.849964	0.846777	0.832644	0.86802	0.781588	0.629482	0.751446	0.581738	0.685764
XGBoost	Best roc_auc	{'learning_rate': 0.1, 'max_depth': 10	0.971455	0.971657	0.971455	0.964829	0.978583	0.768906	0.546125	0.688689	0.582677	0.513889
XGBoost	Best recall	{'learning_rate': 0.05, 'max_depth': 8	0.869446	0.871891	0.869446	0.855861	0.888533	0.782997	0.62439	0.746405	0.587156	0.666667
XGBoost	Best f1	{'learning_rate': 0.1, 'max_depth': 8,	0.89886	0.90025	0.89886	0.88804	0.912802	0.781588	0.602224	0.727964	0.593592	0.611111
FFNN	RandSearchCV	{'optimizer': 'SGD', neurons': 128, lear	0.7556	0.7244	0.7556	0.8306	0.6422	0.7839	0.5922	0.7197	0.6051	0.5799
FFNN	RandSearchCV v	{'optimizer': 'Adam', 'neurons': 16, 'lear	0.7125	0.6562	0.7125	0.8162	0.5486	0.7886	0.5794	0.7099	0.6275	0.5382

Here you can see that the decision tree with the highest recall was also the model that had the lowest accuracy.

Model Evaluation

Best models :

Nonlinear SVM

Runner-up is split between Logistic Regression and Linear SVM

Notes of interest:

Decision tree recall the best but awful accuracy

Other model results non-deterministic

Never beat best models but fluctuated in ranking depending on run

Feature importance from Random Forest

{City development index, training hours, City 21, years of experience}

Our model results were also non-deterministic. We did several runs of models as we cleaned up code and added more details that we wanted to keep track of. Our top performing models tended to stay roughly the same, but the scores themselves definitely fluctuated and the middle and lower rankings of the models would change.

One of the things that we added that we wanted to keep track of was feature importance, based on our random forest models. We had different random forest models but they all agreed on some of the most important features, namely: the city development index value, the number of training hours someone has, the years of experience they have, and whether they live in City 21 or not. We don't actually know what city 21 is but we think it might be San Francisco; people really want to leave their jobs in City 21.

Subgroup Recall

Gender (grouped into male + non-male)

Education level (grouped into college and above vs. high school + primary)

City (City 21 vs. all other cities)

We were also interested in learning about how well our models worked on subgroups as well. We were really wary of how algorithms in HR are being used more and more and could be exacerbating existing inequalities. For example, if our model was used to predict people who might leave so that we could better retain them, if the result was that we were better able to retain men than women, that would exacerbate the tenure difference between genders. We also cared about education level as a proxy for socioeconomic class, and from a more business standpoint, we cared about how our model was performing localized to particular cities, since a model that works well for only one city isn't going to work well for companies in other cities.

Subgroup Recall

Gender (grouped into male + non-male)

Non-male had a recall difference of **+10%** (ranged from **3% to 15%**)

Education level (grouped into college and above vs. high school + primary)

High school + primary had a recall difference of **-14%** (ranged from **-4% to -27%**)

City (City 21 vs. all other cities)

Other cities had a recall difference of **-39%** (ranged from **-30% to -60%**)

We found that the non-male subgroup actually had a better recall of about 10% in our model. Our sample size here is pretty small; we only had around 650 non-male samples, which made up about a third of our test group. We specifically chose to go with non-male instead of female because the female subgroup was even smaller, and there was an option for “Other” under gender and also some people didn’t fill out that field.

Subgroup Recall

Gender (grouped into male + non-male)
Non-male had a recall difference of **+10%** (ranged from **3% to 15%**)

Education level (grouped into college and above vs. high school + primary)
High school + primary had a recall difference of **-14%** (ranged from **-4% to -27%**)

City (City 21 vs. all other cities)
Other cities had a recall difference of **-39%** (ranged from **-30% to -60%**)

Those without a university education had a worse recall of about 14% in our model. We had even fewer samples for this subgroup: only 12%, about 250 of our samples, had at most a primary or high school education.

Subgroup Recall

Gender (grouped into male + non-male)
Non-male had a recall difference of **+10%** (ranged from **3% to 15%**)

Education level (grouped into college and above vs. high school + primary)
High school + primary had a recall difference of **-14%** (ranged from **-4% to -27%**)

City (City 21 vs. all other cities)
Other cities had a recall difference of **-39%** (ranged from **-30% to -60%**)

Also, if you remember city 21, the city where more than 60% of the survey respondents wanted to find a new job, we wondered if this city was skewing the result for all the other cities. Basically, were we localizing too much to city 21? And the answer seems to be yes; there's a pretty big recall gap, where other cities have a lower recall by about 40% across our models. This is actually a pretty big deal because city 21 only makes up 15% of our dataset.

Our Conclusions

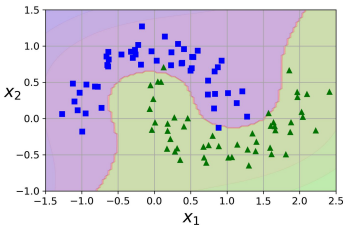
And The Winner Is...

And with the judging and evaluation completed, we now come to the end.

What We've Learned

The Top Model
Non-Linear Support Vector Machines

Everything We Learned in Class is True 🙃
Reached similar conclusions as learned in course



Kaggle Isn't Klean
Many Kaggle submissions artificially inflated model scores through manipulation

DevOps Matters!
Need consistent dev environments, shared libraries, version control

So we crowned our top model: Non-Linear Support Vector Machines. Separating data with a non-linear decision boundary resulted in the best F1 and recall scores by a long shot.

Along the way, we also learned a lot about machine learning.

Firstly, we reached similar conclusions as learned in course - for example, KNN doesn't do well in higher dimensional spaces.

Secondly, we learned that Kaggle is like the Tour Du France: it's full of people doping to win. Some of the top Kaggle submissions achieved higher scores by treating categorical and ordinal data as numeric and throwing out any statistical sense. We were horrified by this…

Lastly, consistent development environments and version controls are very important for workflows! Most of the work-stopping issues we ran into weren't math, concept, or model related. It was figuring out how our dev environments differed and why that meant we couldn't run the same set of code.

The Next Generation

Technical:

CV folds for more generalizability

Thresholds and Confidence Intervals (CIs)
    CIs >>> point estimates

Further subgroup exploration

Business:

Refine data collection on gender, education, etc.

Collect additional features
    Salary, vacation days, etc.

Subgroup by industry, department, and role

City-based models
    Geographic effects *very* strong

Amy - While we learned a lot during this project, there was still a lot more we could have done to improve our results if we had more time. On a technical side, we talked about how the scores were nondeterministic, so we were interested in adding things like CV folds, so that we could get confidence intervals instead of a single score every run. We also wanted to explore more subgroups to delve further into our models and data.

Ryan -
If we had an opportunity to change things at the business level, our models could have been so much more powerful if we had the ability to change aspects of data collection and collect additional data and features.

But hey, maybe the company wouldn't let us do what we wanted. That's okay. There is certainly a company out there that will let us do what we want to do. And as eager, adaptable, mercenary-minded data scientists, we can always change jobs and work for them. Because that is the dream.

Special Thanks

Professor Santerre
For inspiring and guiding us in this project

Hands-on Machine Learning Textbook

The Internet

Our Computers
Sorry for making you work so hard 😭

Special thanks to everyone and everything on this screen. We could not have completed this project without you.

Thank You

Team 1:
Max Eagle, Amy Ho, Ryan Wong

MIDS, Spring 2022
W207, Santerre

Thank you all very much for your time and attention. We would be more than happy to answer any questions you may have.

Appendix

Featured Fields

Demographics

- Gender

City

- City
- City Development Index

Education

- Education Level
- Major Discipline

Employment

- # Years Experience
- # Years since last new job

Current Company

- Company Size
- Company Type

The features included a wide swath of interesting characteristics about the person in question. These features included information about the person's demographics, the city they currently reside in, their education, their employment experience, and their current company.

Target:
“Is this person going to change jobs?

1 = Yes

~75% of entries
responded with Yes.

0 = No

~25% of entries
responded with No.

And at the end of that survey was the ultimate question that was the target of our data set: “Is this person going to change jobs?” And as our baseline, we found that about 75% of the respondents indicated that they were interested in changing jobs, while the other 25% indicated that they were not interested in changing jobs.

Data Pipeline