Loan Default Predictive Modeling

A Case Study of Classification Algorithms

Overview

· Exploration of binary classification models within context of loan default prediction

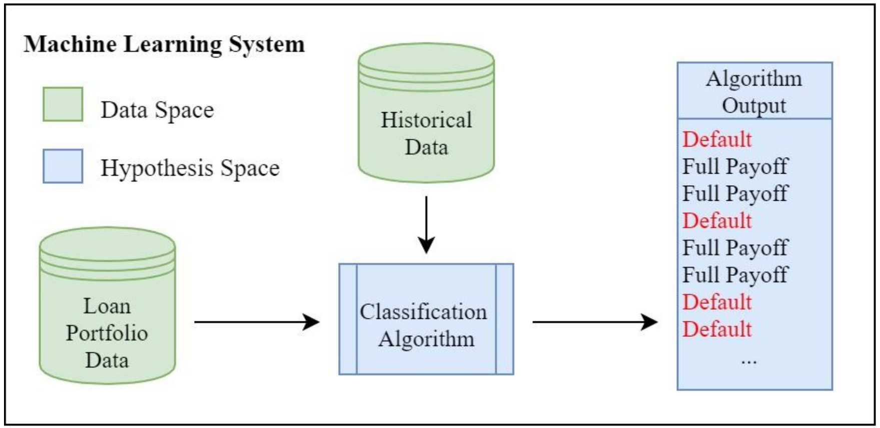
· Evaluate techniques to vary model architecture within *model space*

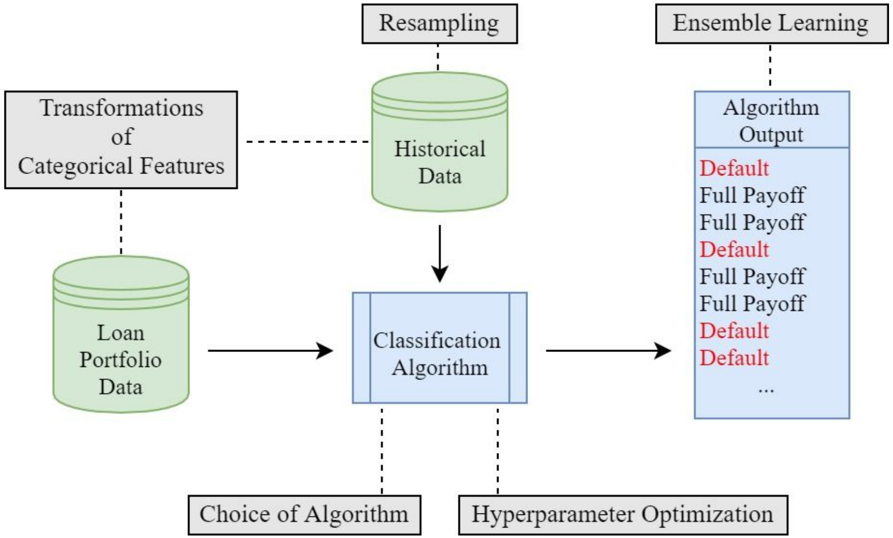
· Techniques operate within two distinct dimensions of *model space*:

- *Data Space*
- *Hypothesis Space*

· Analysis, modeling, and data management conducted with Python, SQL, Excel

Overview - Model Architecture





· Grey: techniques explored in this project to vary model architecture (i.e. particular machine learning system) within *model space*

Data Space

· Includes training data used to fit models as well as features chosen to represent loans

· Two approaches explored to vary model architecture within these aspects of data space:

- Resampling of training data
- Transformations of categorical features

· Resampling: undersampling, SMOTE, ADASYN

· Categories: transformation to indicator variables, application of clustering algorithms

Hypothesis Space

· Includes underlying learning algorithms which map input data to outputs

· Three approaches explored to vary model architecture within hypothesis space:

- Variation of classification algorithms
- Hyperparameter optimization within classification algorithms
- Ensemble Learning Architectures

· Learning algorithms: logistic regression, Gaussian Naive Bayes, k-NN, random forest

· Hyperparameter optimization conducted with grid search cross validation

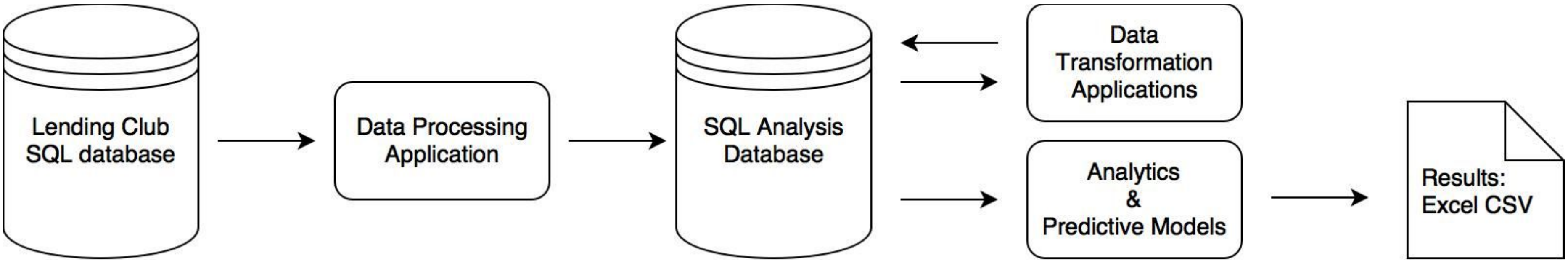
· Ensembles: majority vote, joint probabilities, stacked learning

Data Set

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc	...	last_credit_pull_d	collections_12
0	5000.0	5000.0	4975.0	36	10.65	162.87	6	10	RENT	24000	...	Jan-2016	
1	2500.0	2500.0	2500.0	60	15.27	59.83	5	0	RENT	30000	...	Sep-2013	
2	2400.0	2400.0	2400.0	36	15.96	84.33	5	10	RENT	12252	...	Jan-2016	
3	10000.0	10000.0	10000.0	36	13.49	339.31	5	10	RENT	49200	...	Jan-2015	
5	5000.0	5000.0	5000.0	36	7.90	156.46	7	3	RENT	36000	...	Sep-2015	

- Consumer loan data released by Lending Club
- 880,000 data points and 70+ features representing information about loan and debtor
- Project conducted over curated subset of 250,000 loans and 20 features
- Missing data filled with simple recommendation engine

Data Management



· Pipeline organized for ease of analysis and preservation of data transformations

Data Set Characteristics

- Imbalance of classes - approximate 1:5 ratio of defaulted loans to fully paid loans

- Large degree of feature space overlap between classes

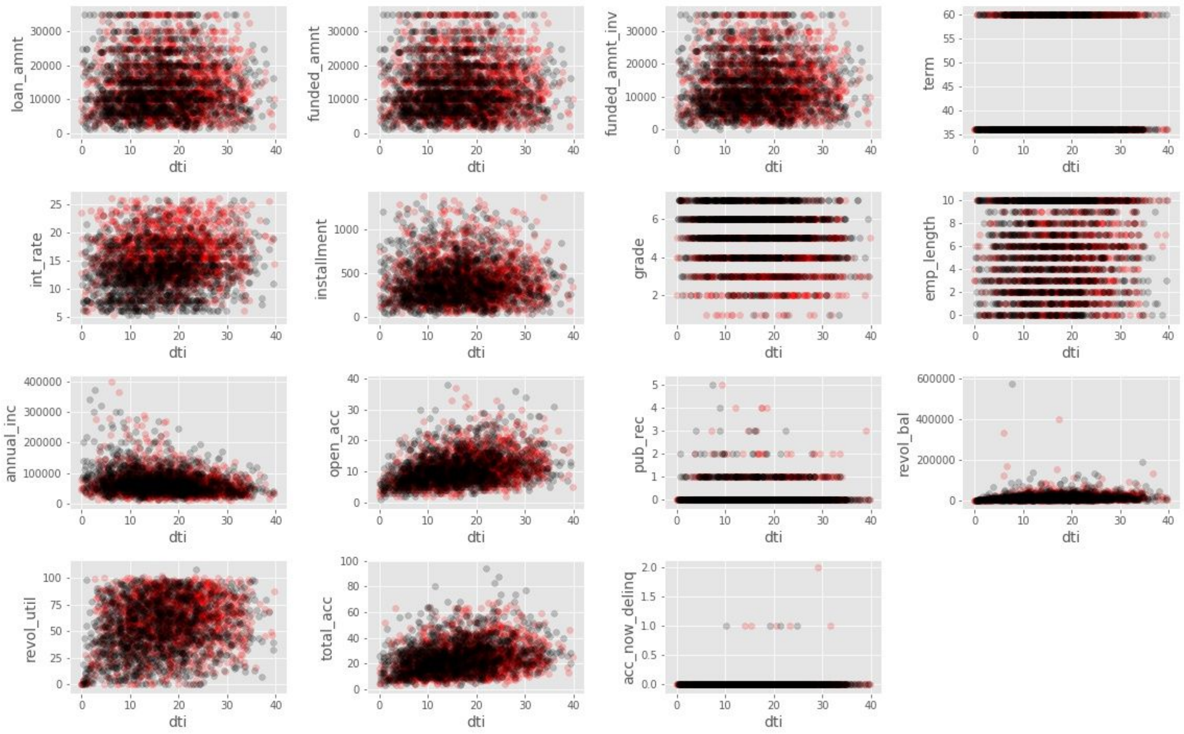
- Causes for poor recall and precision in baseline models (respectively)

Feature Space Overlap

· Random samples of each class (defaults in red)

· Numerical features plotted against debt-to-income ratio

· Note lack of separation between classes for most features

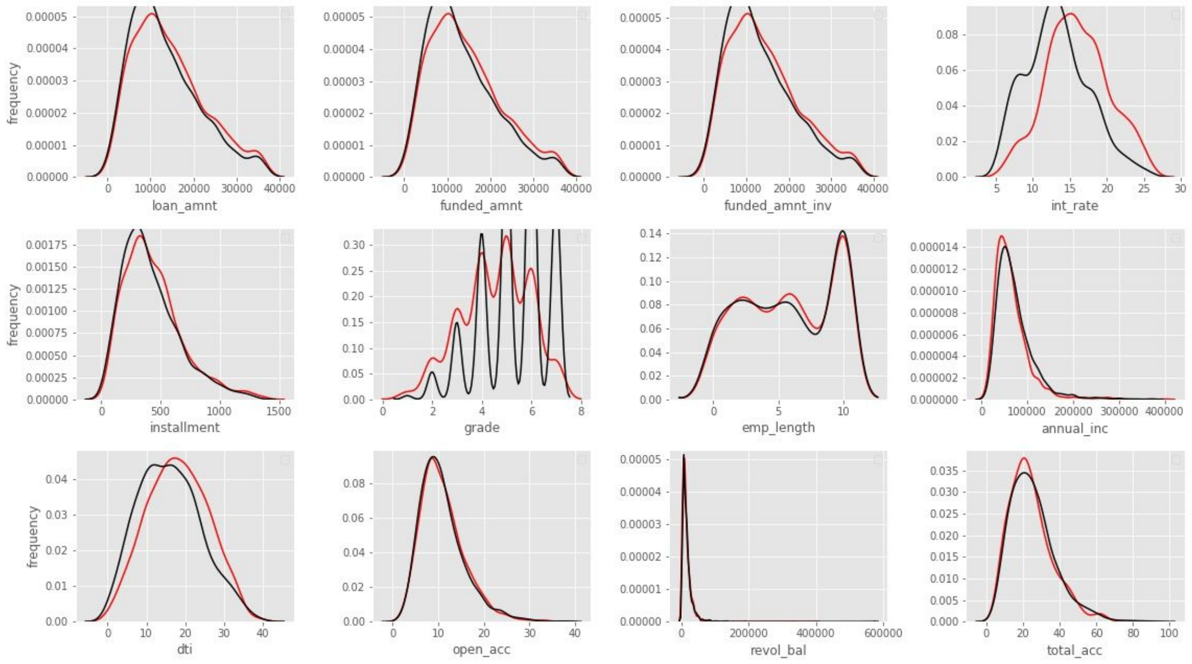


Feature Space Overlap, continued

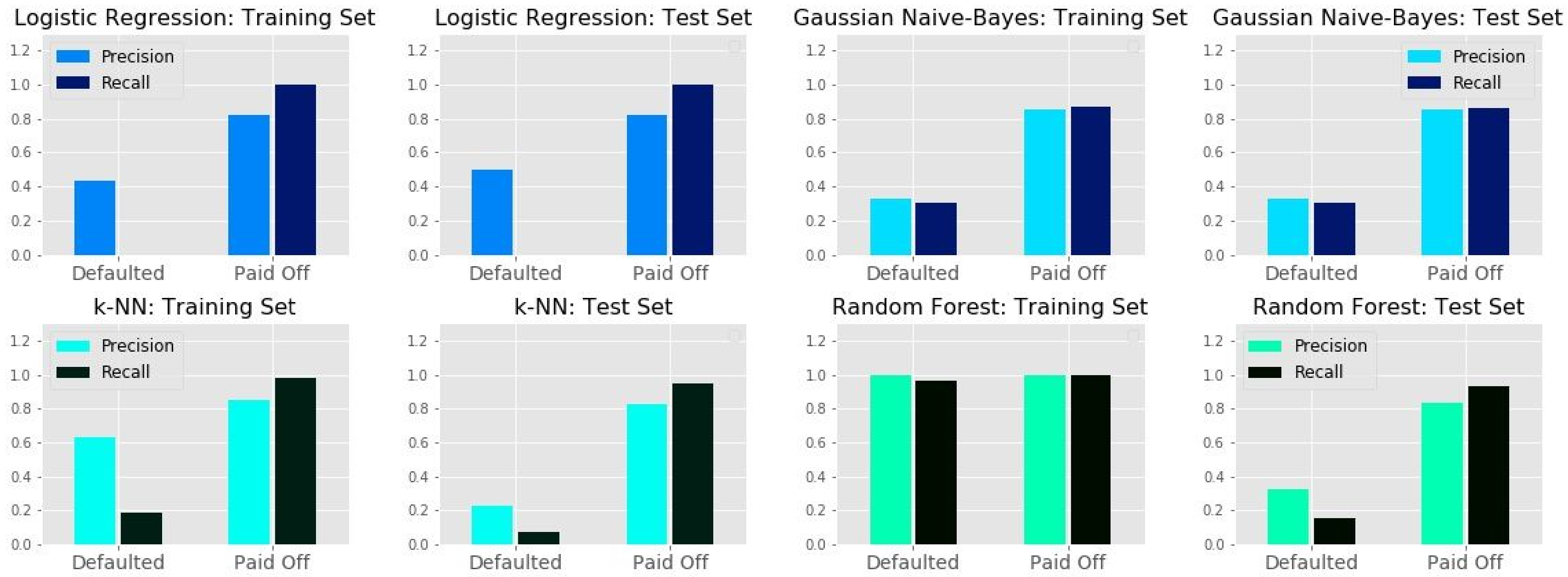
· Random samples of each class (defaults in red)

· Kernel density estimation plots: distribution approximations

· Note lack of significant distinction between classes for most features



Baseline Model Evaluations



- General poor performance on defaults, particularly on recall
- Random forest successfully learns irregular patterns on training set (overfitting)

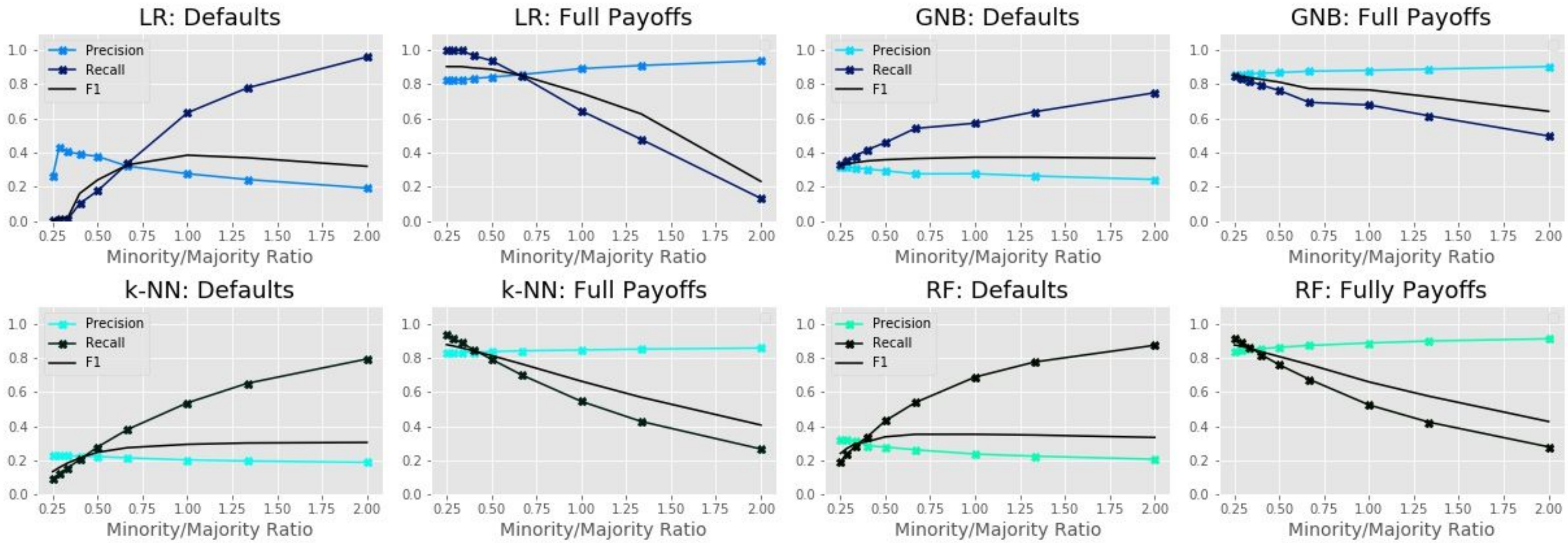
Resampling

· First approach to change model architecture, aimed at improving default recall

· Resampling applied only to training data used in fitting learning algorithms

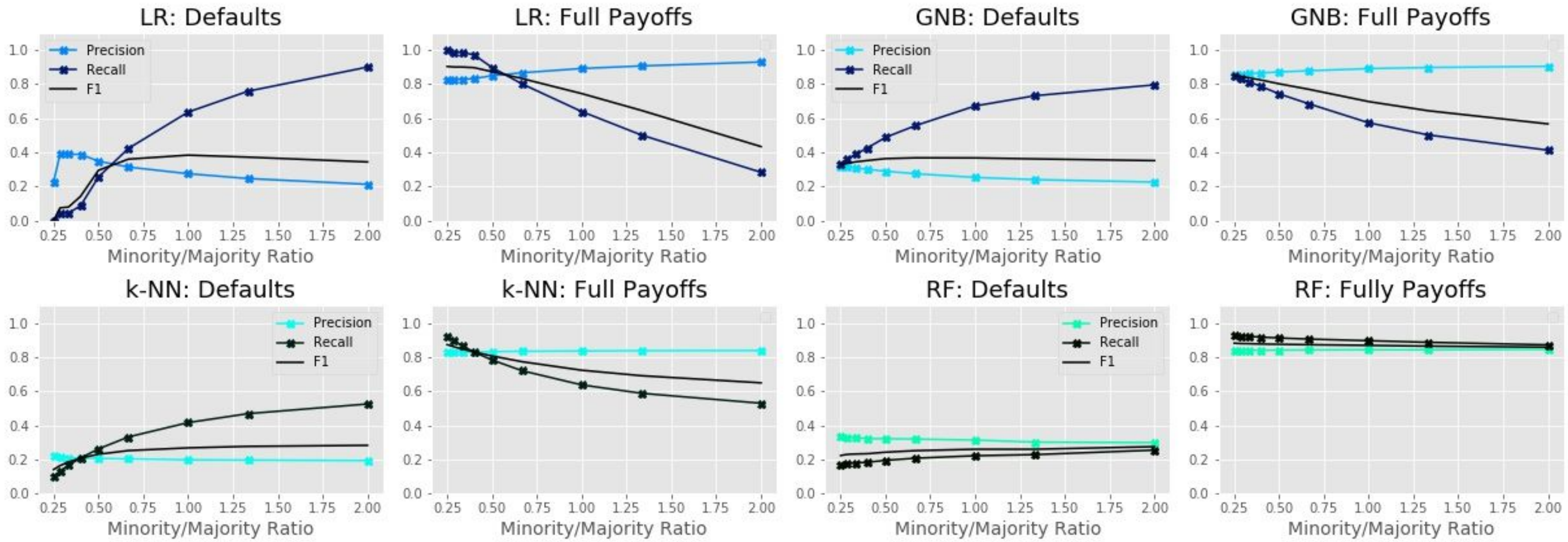
· Want distribution of test set to mimic what is seen in practice, so train-test splits are performed BEFORE resampling on training data

Resampling - Undersampling (results on test set)



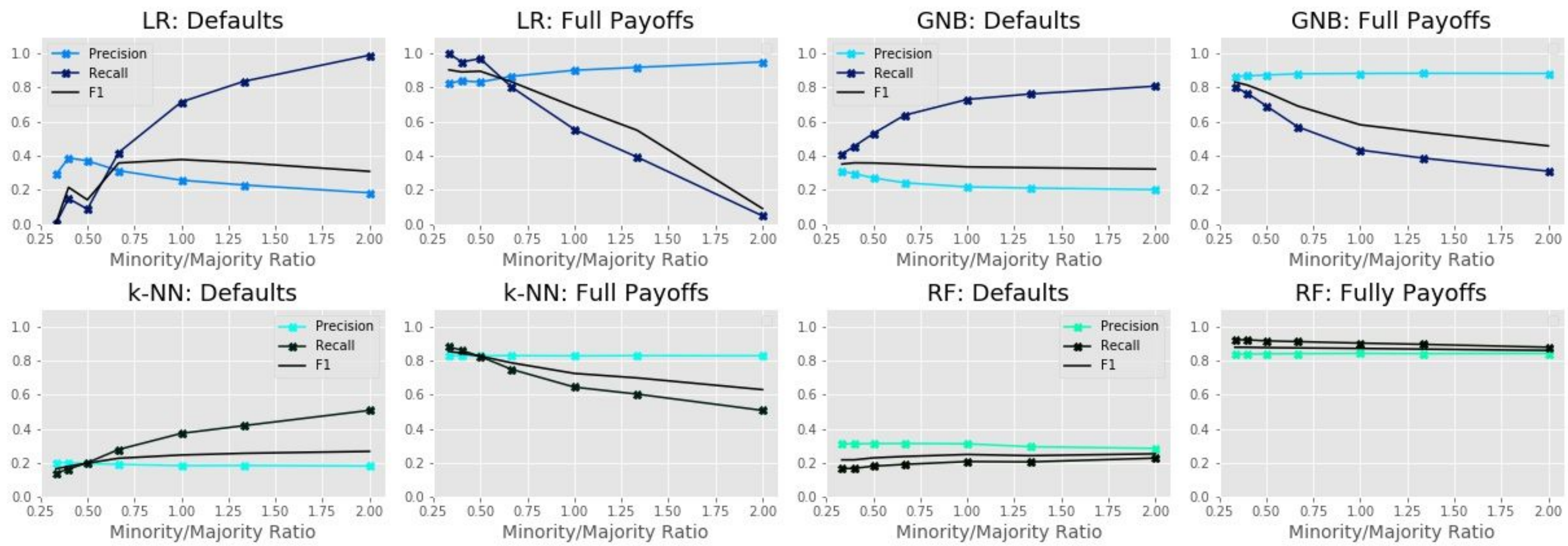
- Undersample majority class for various minority to majority ratios on training set
- Generally enables increases in recall (sensitivity) at expense of recall in opposing class

Resampling - SMOTE (results on test set)



- Similar set of patterns, but models are affected to different extents
- Effects on random forest muted due to SMOTE's approach in generating synthetic data

Resampling - ADASYN (results on test set)



· Overall effects are very similar to SMOTE, due to ADASYN's underlying similarity to SMOTE

Resampling - Conclusions

· Regardless of technique used, there exists trade off of recall between classes

· Precision affected to much lesser extent (generally in opposite direction of recall)

· To apply resampling techniques most effectively, must consider nature of underlying algorithms -- e.g. SMOTE/ADASYN ineffectiveness on random forest

Hyperparameter Optimization

· Aim for general performance improvements by changing hyperparameters of respective models

· Focus is on illustrating grid search cross validation; exhaustive hyperparameter optimization not conducted for lack of computational resources

· Requires case-by-case analysis of algorithms, hyperparameters, and data

· Initially, no resampling applied together with hyperparameter optimization

Logistic Regression: C

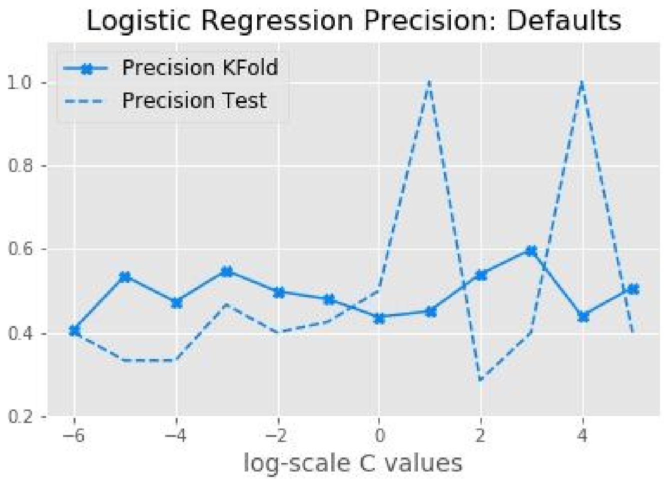
L2: $\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$

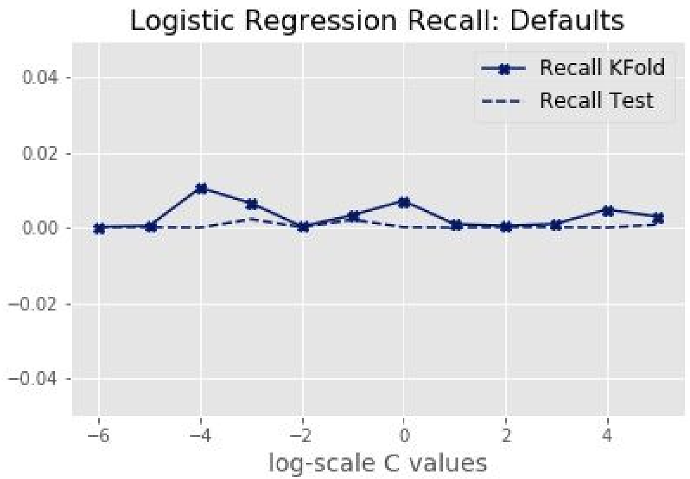
$$J(\mathbf{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi(z^{(i)})\right) - \left(1-y^{(i)}\right)\log\left(1-\phi(z^{(i)})\right)\right] + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

· C: inverse of L2 regularization strength (λ) included in logistic regression loss function

Image: https://www.kdnuggets.com/2016/06/regularization-logistic-regression.html

Logistic Regression: C





· No clear performance trends, likely because logistic regression is so insensitive to predicting defaults when fitting on non-resampled training data

· Recall is impacted to a much lesser extent than precision (note scales of y-axes)

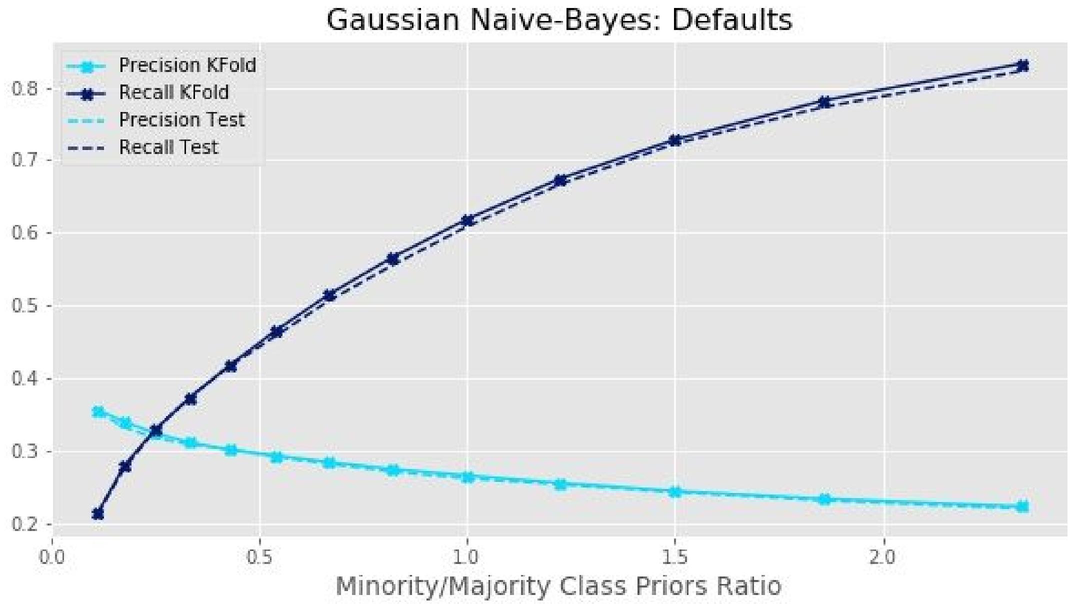
Gaussian Naive Bayes: Prior Probabilities

· Prior probabilities are automatically estimated according to training data, when not specified as arguments

· By specifying specific prior probabilities of each class, can alter tendency of model to output defaults or payoffs

· Thus, expect performance to change in fashion very similar to when applying resampling

Gaussian Naive Bayes: Prior Probabilities



· Indeed, performance characteristics with respect to minority:majority class priors ratio are very similar to those with respect to minority:majority class resampling ratios

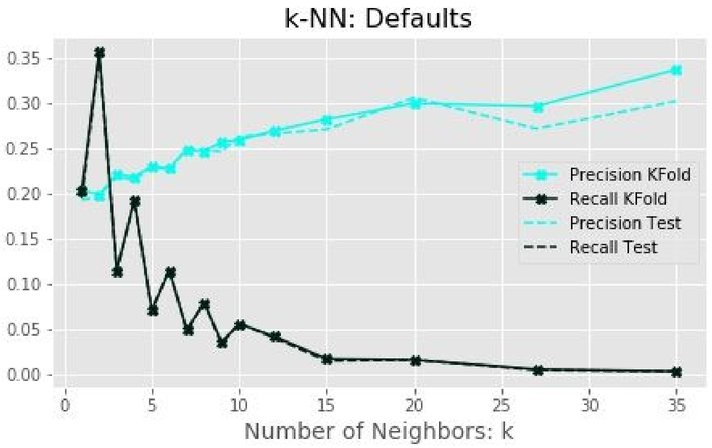
k-NN: Number of Neighbors (k)

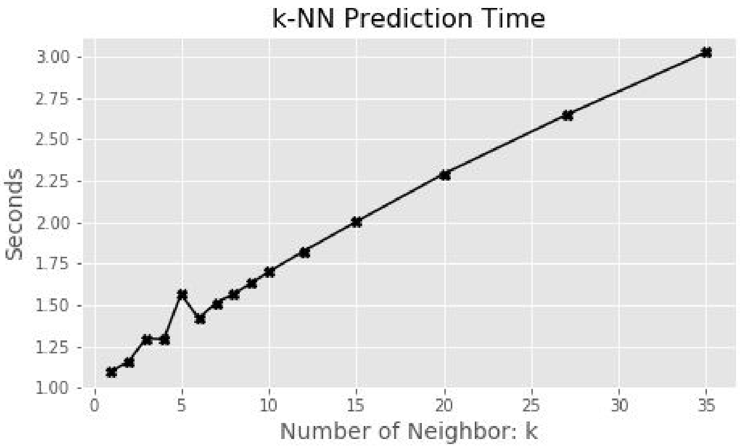
· Number of nearest neighbors (with respect to Euclidean distance metric) used to classify new data points

· Expect general performance increase as k is increased, but possible loss in recall since stricter criteria for prediction

· Performance increase comes at computational expense - must consider increases in prediction time against increases in model performance

k-NN: Number of Neighbors (k)

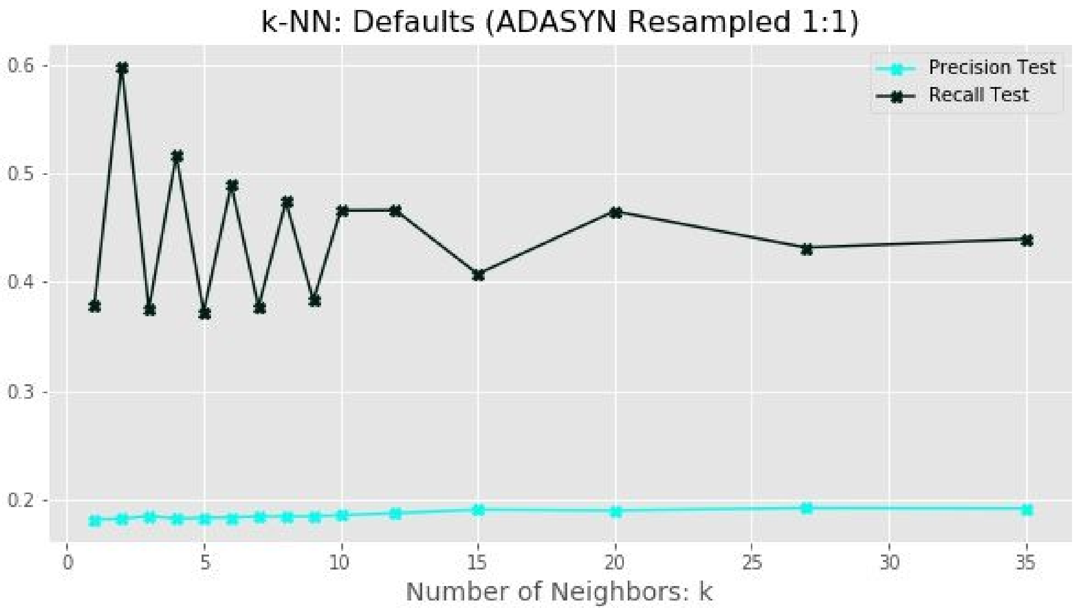




· Decreasing recall likely due to imbalance of classes and feature space overlap

· Precision bounded above at ~ 0.35 (not fully shown) - *k* greater than 35 not worth tradeoff on prediction time

k-NN: Number of Neighbors (k), Resampled



- After resampling, recall no longer decreases (overall) with respect to increasing *k*
- Performance gains in precision become muted

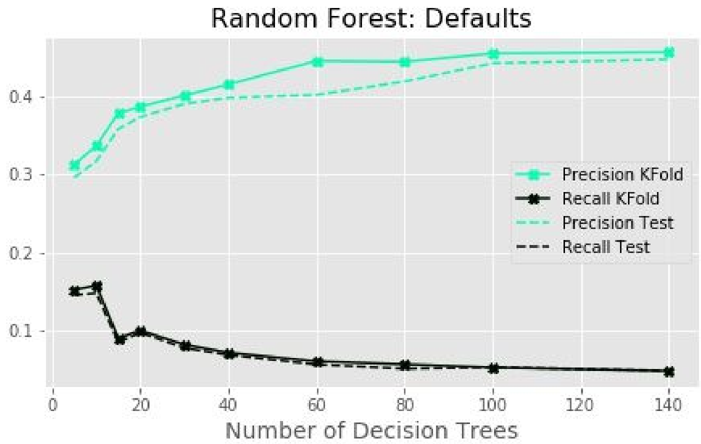
Random Forest: Number of Decision Trees

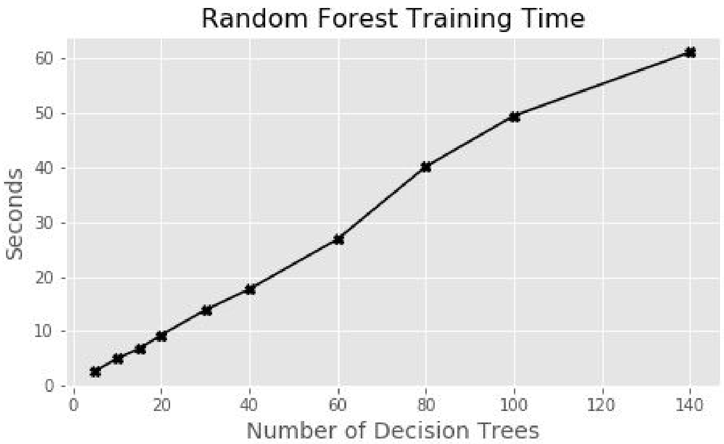
· Number of decision trees used in ensemble

· Expect general performance increase as count of decision trees is increased, but again must consider computational expense

· *Grid search over bigger hyperparameter space is available in Jupyter Notebook - still not exhaustive, however*

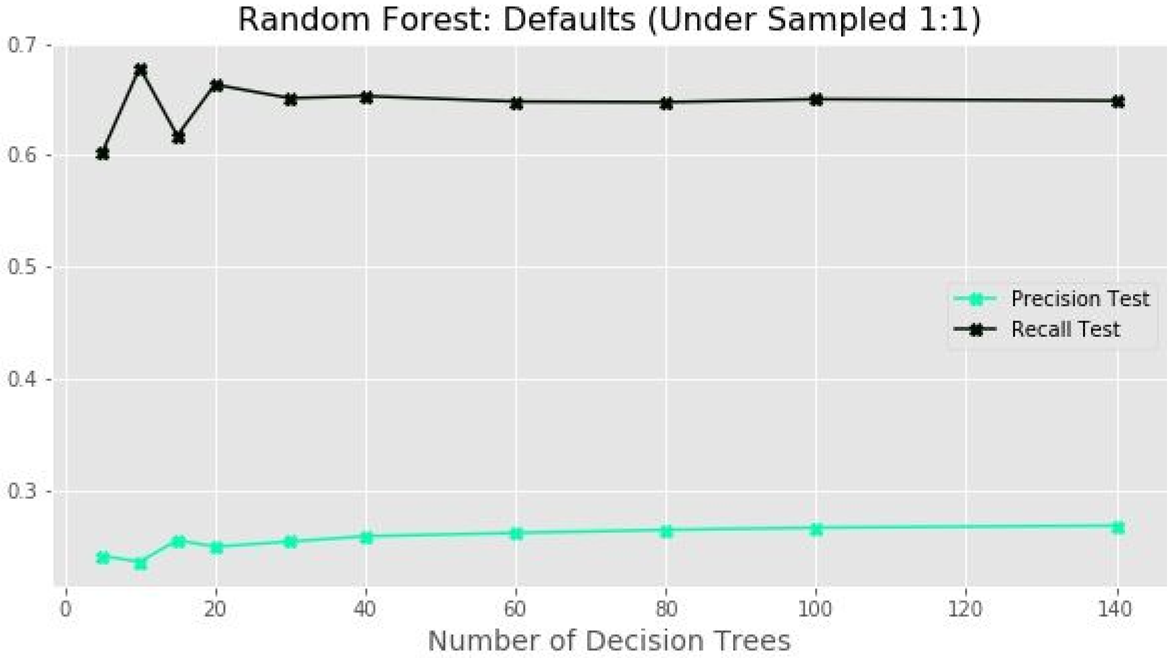
Random Forest: Number of Decision Trees





- Similar performance characteristics as k-NN: increasing precision and decreasing recall
- Precision again looks bounded above (~0.5) with linear computational complexity

Random Forest: Number of Decision Trees, resampled



· Undersampling changes the performance relationship in similar fashion as ADASYN does for k-NN

Hyperparameter Optimization - Conclusions

· Exhaustive explorations of hyperparameter spaces are not complete - need more computing power

· Can alter performance relationships with respect to hyperparameters by first applying resampling, but requires intelligent analysis of algorithms

· Gains in performance are generally bounded with respect to hyperparameter values, due to nature of input data

Pipelining

· Want approach to explore model architectures' performance characteristics more exhaustively and succinctly within model space

· Define Python function to apply resampling techniques and hyperparameter optimization in various combinations

· Method is illustrative of approach which can be generalized to explore more variations of model architecture

Pipelining - Default Class Results

model	N_pre	N_rec	UnderSamp_N_pre	UnderSamp_N_rec	SMOTE_N_pre	SMOTE_N_rec	ADASYN_N_pre	ADASYN_N_rec
LR	0.500000	0.000224	0.273209	0.640721	0.273460	0.630866	0.253973	0.726733
reLR	0.500000	0.000224	0.273283	0.640497	0.274015	0.634786	0.254445	0.719677
GNB	0.325714	0.307985	0.266798	0.607011	0.252503	0.669392	0.217562	0.726173
reGNB	0.263459	0.608915	0.256094	0.664800	0.236426	0.750028	0.208146	0.788106
KNN	0.229395	0.070445	0.203121	0.543734	0.196971	0.413596	0.183698	0.373054
reKNN	0.215069	0.191175	0.194014	0.706350	0.195092	0.370366	0.183156	0.517303
RF	0.335203	0.160600	0.238553	0.688543	0.311747	0.223205	0.304969	0.210326
reRF	0.331065	0.152537	0.259314	0.664128	0.314198	0.207190	0.311830	0.169448

· Grid Search CV optimized models: ‘*re-*’

· All resampling applied with 1:1 ratio

· Precision: ‘*pre*’ ; Recall: ‘*rec*’

Pipelining - Payoff Class Results

model	P_pre	P_rec	UnderSamp_P_pre	UnderSamp_P_rec	SMOTE_P_pre	SMOTE_P_rec	ADASYN_P_pre	ADASYN_P_rec
LR	0.823123	0.999952	0.891379	0.633674	0.889674	0.639764	0.902102	0.541196
reLR	0.823123	0.999952	0.891360	0.633939	0.890530	0.638537	0.900749	0.546781
GNB	0.852989	0.862968	0.883650	0.641473	0.889863	0.574100	0.881718	0.438705
reGNB	0.882964	0.634132	0.890346	0.584956	0.899223	0.479384	0.886475	0.355614
KNN	0.826112	0.949139	0.846681	0.541533	0.834957	0.637598	0.826907	0.643712
reKNN	0.830218	0.850042	0.854058	0.369334	0.832294	0.671585	0.829341	0.504152
RF	0.837757	0.931544	0.887418	0.527645	0.842653	0.894091	0.840893	0.896979
reRF	0.836777	0.933759	0.891364	0.592298	0.841228	0.902804	0.837447	0.919629

- Corresponding performance levels on positive class (must consider tradeoffs)
- More exhaustive study with varying resampling ratios would results in multiple tables

Pipelining - Conclusions

· Pipelining enables succinct evaluation of model architectures within model space

· With sufficient computational resources, can explore model space more exhaustively and discover better model architectures

· Underlying model architectures must still be analyzed to make sense of results and effectively construct new architectures

Categorical Features

· Cannot include categories as they are -- algorithms work on numerical inputs (even if categories are numbers, results would be nonsensical)

· Transform categories to indicator features: new feature for each categorical value

· Develop two variations of features with categorical data included and transformed: non-clustered and clustered

· Re-run pipeline with these variations of input features to evaluate performance

Categorical Features - No Clustering

Numerical Features			Categorical Features		
Feature 1	...	Feature 12	Feature 13	...	Feature 20

Numerical Features			Indicator Features		
Feature 1	...	Feature 12	Feature 13	...	Feature 977

Numerical Features			Indicator Features		
Feature 1	...	Feature 12	Feature 13	...	Feature 977

Numerical Features			PCA-Reduced Indicator Features		
Feature 1	...	Feature 12	Feature 13	...	Feature 24

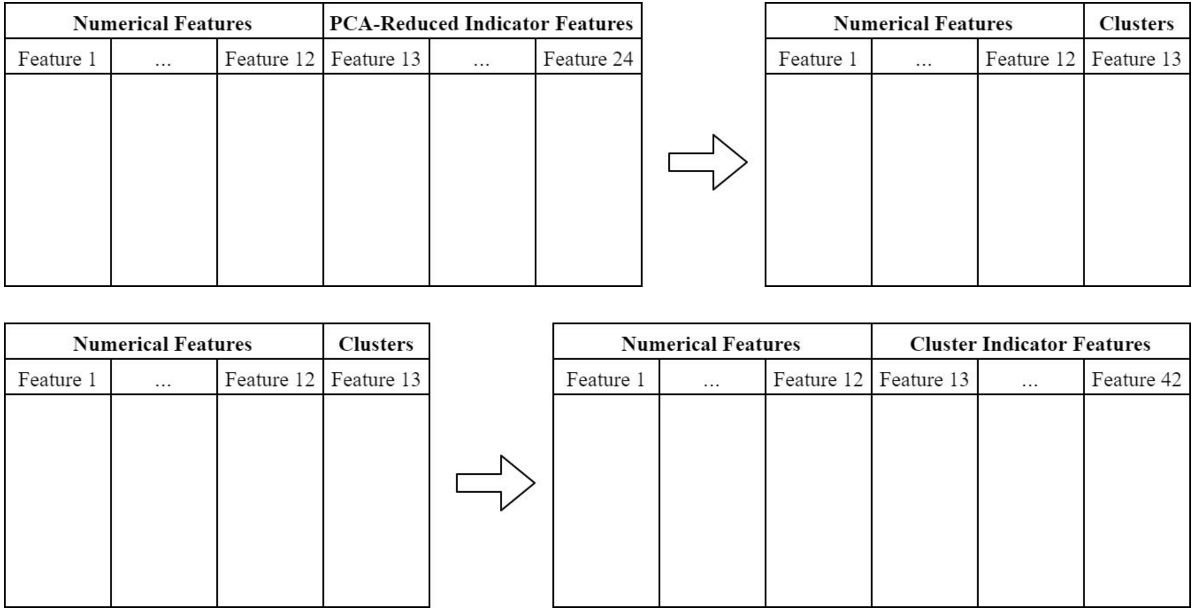
· Indicator transformation greatly increase feature space dimension - apply PCA

Categorical Features - No Clustering Results (Defaults)

model	N_pre	N_rec	UnderSamp_N_pre	UnderSamp_N_rec	SMOTE_N_pre	SMOTE_N_rec	ADASYN_N_pre	ADASYN_N_rec
LR	0.426554	0.017151	0.274980	0.635166	0.274922	0.633689	0.257353	0.714562
reLR	0.402256	0.012154	0.274669	0.632326	0.271399	0.634939	0.255227	0.715470
GNB	0.318875	0.332349	0.274333	0.566333	0.262404	0.618128	0.231312	0.685711
reGNB	0.261056	0.644366	0.260950	0.642890	0.248248	0.696274	0.220014	0.760904
KNN	0.300030	0.111881	0.230889	0.587347	0.225715	0.508065	0.244896	0.290209
reKNN	0.265069	0.251249	0.208885	0.733303	0.223567	0.448887	0.244571	0.283962
RF	0.316195	0.144593	0.230510	0.690482	0.300139	0.196161	0.301639	0.177647
reRF	0.317656	0.149591	0.265471	0.647547	0.347698	0.120968	0.375723	0.066447

· Some model architectures perform better than when only numerical features are used, but many are not - inclusion of categories does not result in strictly better performance

Categorical Features - Clustered (K-Means)



- Clustering results in categorical feature - must apply indicator transformation again

Categorical Features - Clustered Results (Defaults)

model	N_pre	N_rec	UnderSamp_N_pre	UnderSamp_N_rec	SMOTE_N_pre	SMOTE_N_rec	ADASYN_N_pre	ADASYN_N_rec
LR	0.427136	0.019309	0.273490	0.625511	0.271991	0.629373	0.256371	0.708428
reLR	0.404494	0.008178	0.273836	0.625057	0.272186	0.627783	0.256325	0.708882
GNB	0.316061	0.320082	0.270894	0.566220	0.241990	0.727510	0.209674	0.825193
reGNB	0.270347	0.562926	0.256158	0.647319	0.227005	0.800886	0.200532	0.873807
KNN	0.297207	0.111199	0.227424	0.588142	0.223180	0.497160	0.242788	0.286801
reKNN	0.264309	0.251249	0.209091	0.735120	0.223570	0.445820	0.246706	0.287142
RF	0.302279	0.138573	0.227376	0.684348	0.302046	0.199568	0.292909	0.182985
reRF	0.308200	0.137892	0.259681	0.679464	0.328131	0.183894	0.315528	0.108019

- Some model architectures perform better than when categories are not clustered
- Must perform deeper analysis to understand why certain architectures perform better

Categorical Features - Alternative Approach

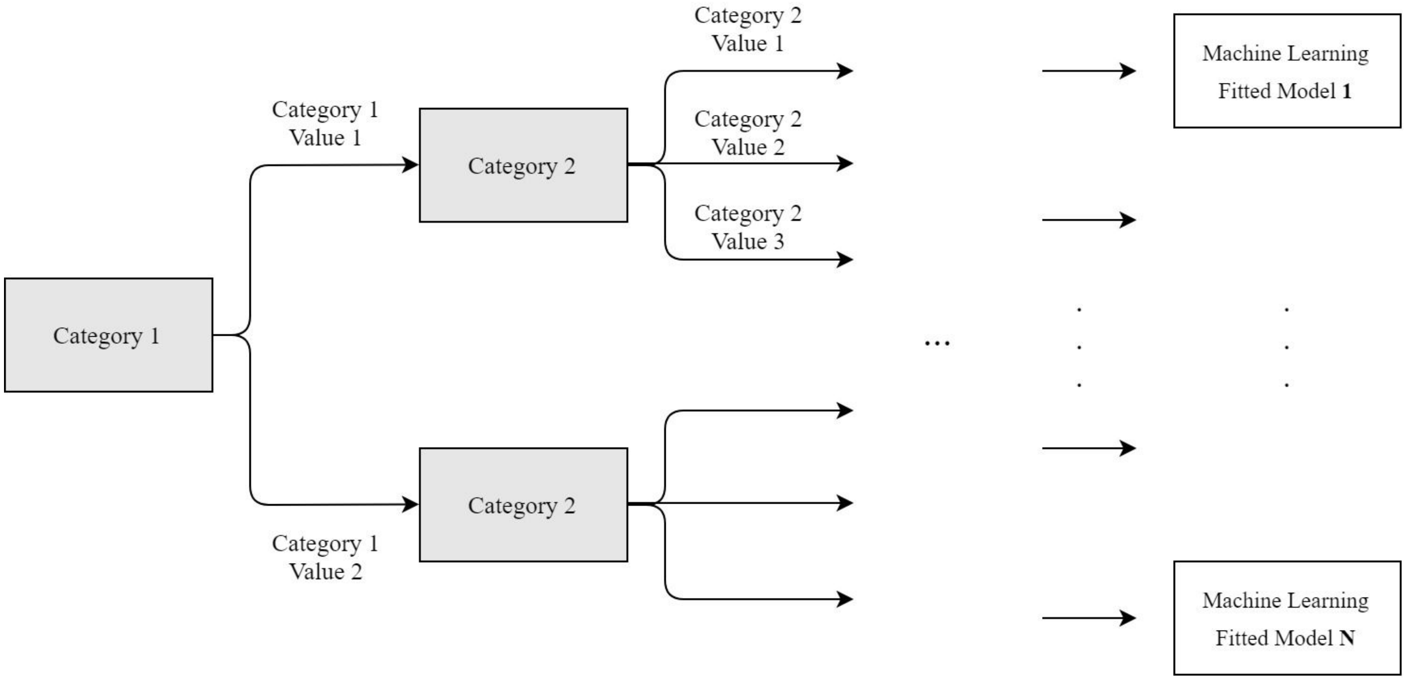
· Develop separate model for each combination of categorical values

· Total time to train models should not be much greater than training single model, since each model would be trained on mutually exclusive subset of data

· Would need to ensure that each combination of categorical values has sufficient training data

· Would need to develop efficient (vectorized) approach to predicting labels for batches of test data

Categorical Features - Alternative Approach



· **N** - number of combinations of unique categorical values across all categories

Categorical Features - Conclusions

· There is much room for creativity in transforming categorical data to forms that can be effectively input to learning algorithms

· Usage of all features (i.e. inclusion of categorical data) does not guarantee better performance

· Again, must analyze underlying algorithms together with input data to construct effective model architecture

Ensemble Learning

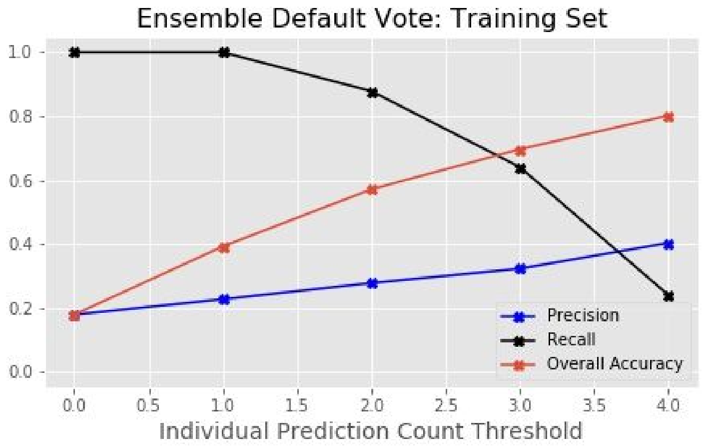
· Explore three methods to combine outputs of learning algorithms to single prediction, i.e. use 'heterogeneous ensembles'

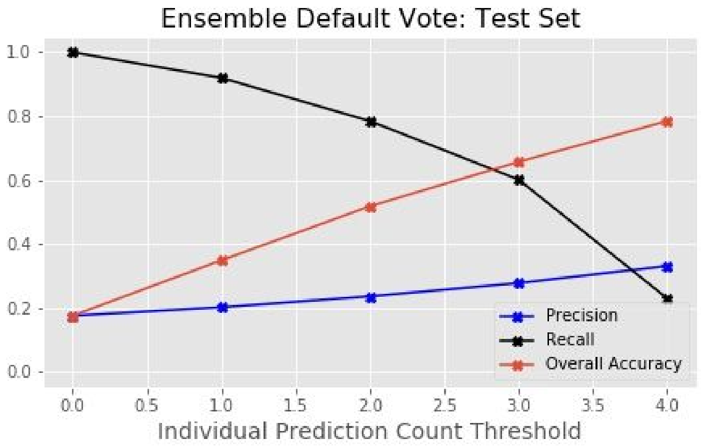
· *'Majority Vote'* - not really majority; vary the threshold vote count for prediction

· '*Joint Probability'* - use product of individual probabilities; vary threshold probability

· '*Stacking'* - run classification algorithms 'in sequence' on initial predicted probabilities

Ensemble Learning - ‘Majority’ Vote

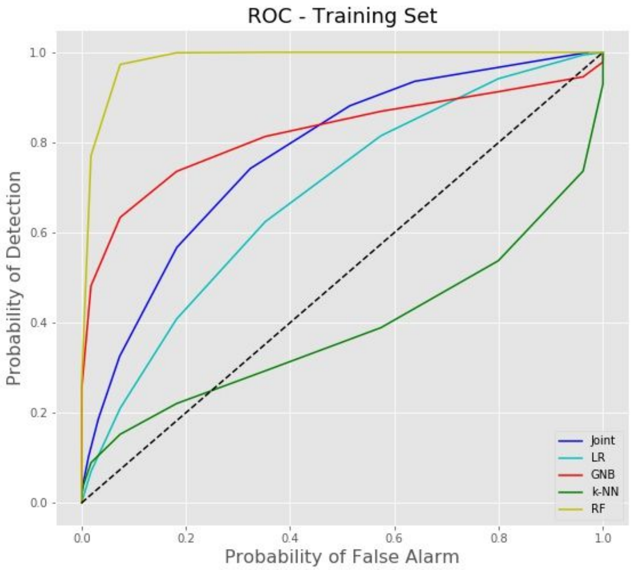


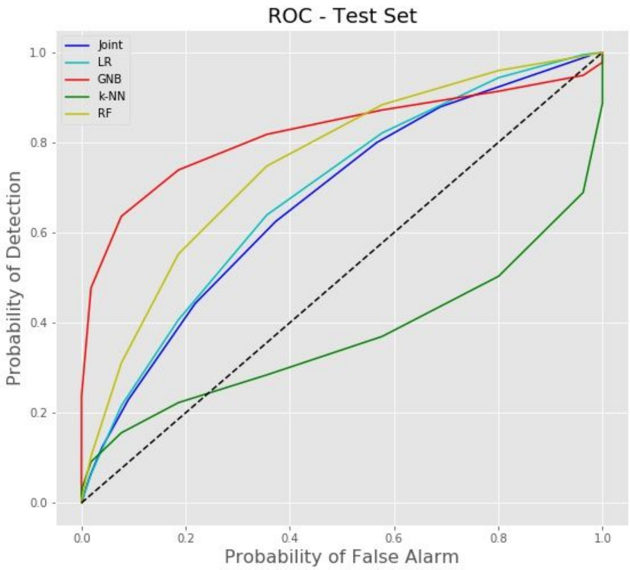


· Vary the count of individual default predictions needed for ensemble to predict default

· More default votes needed corresponds to stricter criteria for default prediction

Ensemble Learning - Joint Probability

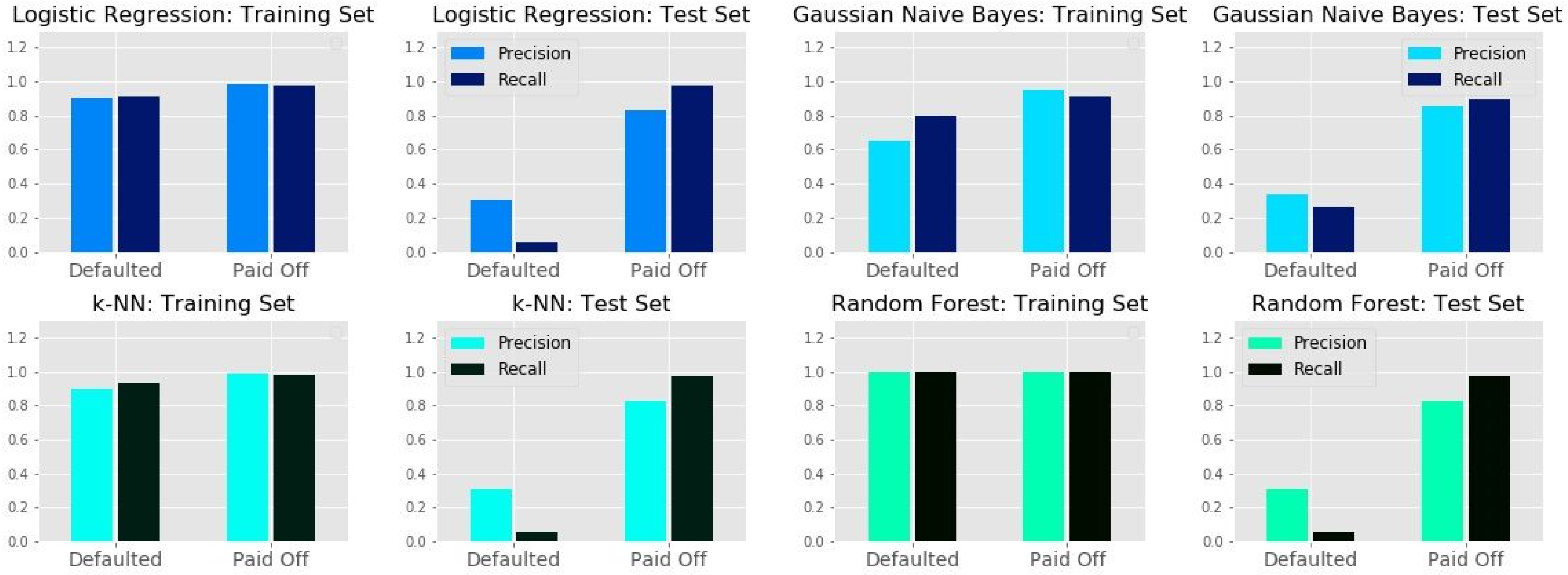




· k-NN's approach to outputting probabilities does not seem well suited in this context

· Using joint probabilities does not seem to give an advantage over using single best

Ensemble Learning - Stacked Classification



- Predicted probabilities of all 4 algorithms are re-input into respective algorithms
- Algorithms find good structure in ‘probabilistic features,’ but do not generalize (overfit)

Ensemble Learning - Conclusions

· There is much room for creativity in developing ensemble architectures

· With many ensemble architectures, the discrimination threshold is an additional parameter which can be altered to achieve different performance levels

· Within the context of this project, ensemble methods do not offer much greater performance levels than using individual algorithms

· Many techniques have not been explored: AdaBoost, usage of more classifiers, etc.

Conclusions - Overview

· Have explored various model architectures within *model space*, but study is not exhaustive

· Focus has been on illustrating approach for systematically evaluating different models

· More exhaustive study would be possible with sufficient computational resources

· To effectively structure model architecture, must analyze underlying algorithms

Conclusions - Client Recommendations

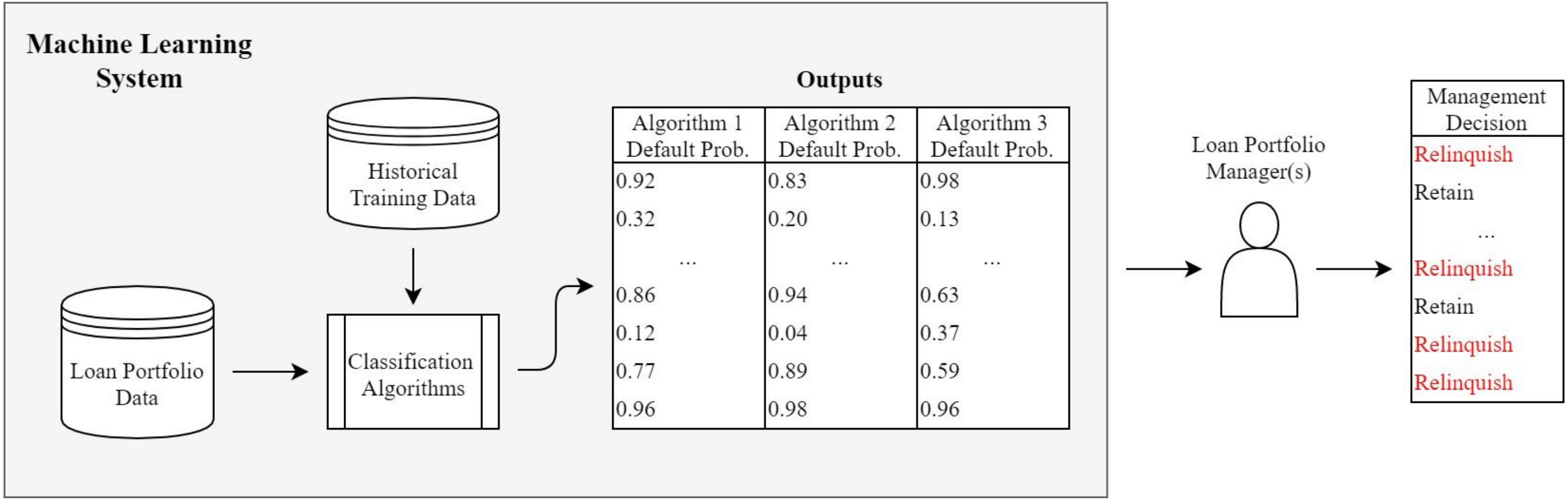
· Bounds on precision are too low for fully automated loan portfolio management

· For better precision, need more relevant features to characterize loans

· System would be more reliable by outputting probabilities rather than predictions

· Should use outputs of machine learning system together with other financial and economic analysis for decision making

Conclusions - Client Recommendations



· Machine learning systems can *aid* in decision making workflows, but with performance levels seen in this project they should not *replace* decision making