Brian Elinsky

Professor Lawrence Fulton, Ph.D.

2020SP_MSDS_422-DL_SEC55

26 April 2020

Assignment 3: Evaluating Classification Models

PROBLEM DESCRIPTION

Given a set of explanatory variables for titanic passengers, my business objective is to predict whether or note each passenger will survive. In this fictional scenario, I am providing evidence regarding characteristics associated survival to a historian writing a book. This indicates to me that the model does not need to be extremely accurate at predicting survival on an individual basis. Rather it is more important to develop a model that is interpretable, and predicts survival reasonably well.

RESEARCH DESIGN AND MODELING METHODS

My plan was to start by fitting a basic logistic regression model using a 10-fold cross validation methodology. Then fit a simple Naive Bayes classifier. Afterwards, I would select the more promising model, then fine tune it for the final predictions.

DATA PREPARATION

I decided to throw out the following variables: Name, Ticket, and Cabin. I don't think Name or Ticket will have much predictive value. There is a lot of missing data for the Cabin variable. Missing categorical data was imputed with the most frequent values. Missing numeric data was imputed with median values.

For the logistic regression, I encoded categorical variables as dummy variables. For the Naive Bayes classifier, I encoded categorical variables as numbers with an Ordinal Encoder.

RESULTS AND MODEL EVALUATION

Both models performed similarly. The AUC for the logistic regression was 0.849 and for the Bayes Naive classifier it was 0.831. Accuracy for both was 79%. This is a significant improvement over the 38% base rate of survival.

Next, I decided to add in regularization to my logistic regression model. A grid search found that an L2 penalty with C=0.1 performed best, although the improvement was only slightly better than the simpler logistic regression with no penalty.

The learning curves indicated that this model likely under-fitting the training data. The curves quickly reach a high plateau, and are very close to each other. This indicates that a more complex model may improve its predictive capacity.

The Kaggle score for the final logistic regression model was 0.76555. The kaggle score for the final Bayes Naive classifier was 0.75598 (https://www.kaggle.com/brianelinsky) (Account User ID 4810027).

MANAGEMENT RECOMMENDATIONS

Examining the logistic regression coefficients indicates that survivors tended to be younger, richer, or female. However, it is prudent to analyze these coefficients with a grain of salt. The explanatory variables likely aren't independent of each other. The predictive power of one attribute could be encoded in another. A decision tree with a shallow depth would likely make a good enough model with better explainability.

```python
# runall.py
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, precision_recall_curve, roc_curve, auc
from sklearn.model_selection import cross_val_predict, GridSearchCV, ShuffleSplit
from sklearn.pipeline import make_pipeline, FeatureUnion
from sklearn.preprocessing import OneHotEncoder, RobustScaler

from library.library import *

# Load training data
from library.transformers import ColumnSelector, TypeSelector, CategoricalEncoder

train_data_filepath = get_dataset_file_path('2020-04-04', 'train.csv')
train_df = pd.read_csv(train_data_filepath)

# Remove label
X_train = train_df.drop(columns='Survived')
y_train = train_df['Survived'].copy()

# Variables to use in the model
x_cols = ['Pclass',
          'Sex',
          'Age',
          'SibSp',
          'Parch',
          'Fare',
          'Embarked']

# Build pre-processing pipeline
preprocess_pipeline = make_pipeline(
    ColumnSelector(columns=x_cols),
    CategoricalEncoder(),
    FeatureUnion(transformer_list=[
        ("numeric_features", make_pipeline(
            TypeSelector(np.number),
            SimpleImputer(missing_values=np.nan, strategy='median'),
            RobustScaler()
        )),
        ("categorical_features", make_pipeline(
            TypeSelector("category"),
            SimpleImputer(strategy="most_frequent"),
            OneHotEncoder()
        ))
    ])
)

# Preprocess data
X_train = preprocess_pipeline.fit_transform(X_train)

# Fine tune the Logistic Regression model using a randomized search cross validation
param_grid = [
    {"C": [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],
     "penalty": ["l1", "l2"],
     "max_iter": [100, 1000, 1000],
     "solver": ["liblinear"]}
]

grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=10, scoring="roc_auc", n_jobs=4,
                           return_train_score=True)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
```

```
final_model = grid_search.best_estimator_

# Evaluate final model on training data

# Generate cross-validated estimates for each data point
y_train_pred_proba = cross_val_predict(final_model, X_train, y_train, cv=10, method="predict_proba")

# Compute precision and recall for all possible thresholds
precisions, recalls, thresholds = precision_recall_curve(y_train, y_train_pred_proba[:, 1])

# Plot precision vs recall graph
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.savefig('Precision_Recall_plot.svg')
plt.show()

# Generate precision and recall stats
y_train_pred_bool = y_train_pred_proba[:, 1] > 0.5
y_train_pred_binary = y_train_pred_bool.astype(int)
precision_score = precision_score(y_train, y_train_pred_binary)
recall_score = recall_score(y_train, y_train_pred_binary)
print("Precision Score:", precision_score)
print("Recall Score:", recall_score)

# Generate accuracy stats
accuracy = sum(y_train_pred_binary == y_train) / y_train.size
print("Accuracy: ", accuracy)

# Plot the ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_train_pred_proba[:, 1])
plot_roc_curve(fpr, tpr)
plt.savefig('AUC_plot.svg')
plt.show()

# Calculate area under the curve
area_under_curve = auc(fpr, tpr)
print("Area under the curve: ", area_under_curve)

# Plot Learning Curve
fig, axes = plt.subplots(3, 1, figsize=(10, 15))
title = "Learning Curves (Logistic Regression)"
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=34)
estimator = LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                               intercept_scaling=1, l1_ratio=None, max_iter=100,
                               multi_class='auto', n_jobs=None, penalty='l2',
                               random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                               warm_start=False)
plot_learning_curve(estimator, title, X_train, y_train, axes=axes, ylim=(0, 1.01))
plt.savefig('learning_curve.svg')

# Generate final predictions on test data

# Load test data
data_file = get_dataset_file_path('2020-04-04', 'test.csv')
X_test = pd.read_csv(data_file)
output = X_test["PassengerId"]

# Preprocess data
X_test = preprocess_pipeline.transform(X_test)

# Make final predictions
final_predictions = final_model.predict(X_test)

# Output predictions
final_predictions = pd.Series(final_predictions, name="Survived")
output = pd.concat([output, final_predictions], axis=1)
output.to_csv("predictions.csv", index=False)
```

```
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.metrics import precision_score, recall_score, precision_recall_curve, roc_curve, auc
from sklearn.model_selection import cross_val_predict
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import make_pipeline, FeatureUnion
from sklearn.preprocessing import OrdinalEncoder

from library.library import *
from library.transformers import ColumnSelector, TypeSelector, CategoricalEncoder

# Load training data

train_data_filepath = get_dataset_file_path('2020-04-04', 'train.csv')
train_df = pd.read_csv(train_data_filepath)

# Remove label
X_train = train_df.drop(columns='Survived')
y_train = train_df['Survived'].copy()

# Variables to use in the model
x_cols = ['Pclass',
          'Sex',
          'Age',
          'SibSp',
          'Parch',
          'Fare',
          'Embarked']

# Build pre-processing pipeline
preprocess_pipeline = make_pipeline(
    ColumnSelector(columns=x_cols),
    CategoricalEncoder(),
    FeatureUnion(transformer_list=[
        ("numeric_features", make_pipeline(
            TypeSelector(np.number),
            SimpleImputer(missing_values=np.nan, strategy='median'),
        )),
        ("categorical_features", make_pipeline(
            TypeSelector("category"),
            SimpleImputer(strategy="most_frequent"),
            OrdinalEncoder()
        ))
    ])
)

# Preprocess data
X_train = preprocess_pipeline.fit_transform(X_train)

# Instantiate and train a simple Naive Bayes classifier model
nb = GaussianNB()

# Generate cross-validated estimates for each data point
y_train_pred_proba = cross_val_predict(nb, X_train, y_train, cv=10, method="predict_proba")

# Compute precision and recall for all possible thresholds
precisions, recalls, thresholds = precision_recall_curve(y_train, y_train_pred_proba[:, 1])

# Plot precision vs recall graph
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.savefig('Precision_Recall_plot.svg')
```

```
plt.show()

# Generate precision and recall stats
y_train_pred_bool = y_train_pred_proba[:, 1] > 0.5
y_train_pred_binary = y_train_pred_bool.astype(int)
precision_score = precision_score(y_train, y_train_pred_binary)
recall_score = recall_score(y_train, y_train_pred_binary)
print("Precision Score:", precision_score)
print("Recall Score:", recall_score)

# Generate accuracy stats
accuracy = sum(y_train_pred_binary == y_train) / y_train.size
print("Accuracy: ", accuracy)

# Plot the ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_train_pred_proba[:, 1])
plot_roc_curve(fpr, tpr)
plt.savefig('AUC_plot.svg')
plt.show()

# Calculate area under the curve
area_under_curve = auc(fpr, tpr)
print("Area under the curve: ", area_under_curve)

# Generate final predictions on test data

# Load test data
data_file = get_dataset_file_path('2020-04-04', 'test.csv')
X_test = pd.read_csv(data_file)
output = X_test["PassengerId"]

# Preprocess data
X_test = preprocess_pipeline.transform(X_test)

# Fit model
final_model = GaussianNB().fit(X_train, y_train)

# Make final predictions
final_predictions = final_model.predict(X_test)

# Output predictions
final_predictions = pd.Series(final_predictions, name="Survived")
output = pd.concat([output, final_predictions], axis=1)
output.to_csv("predictions.csv", index=False)
```

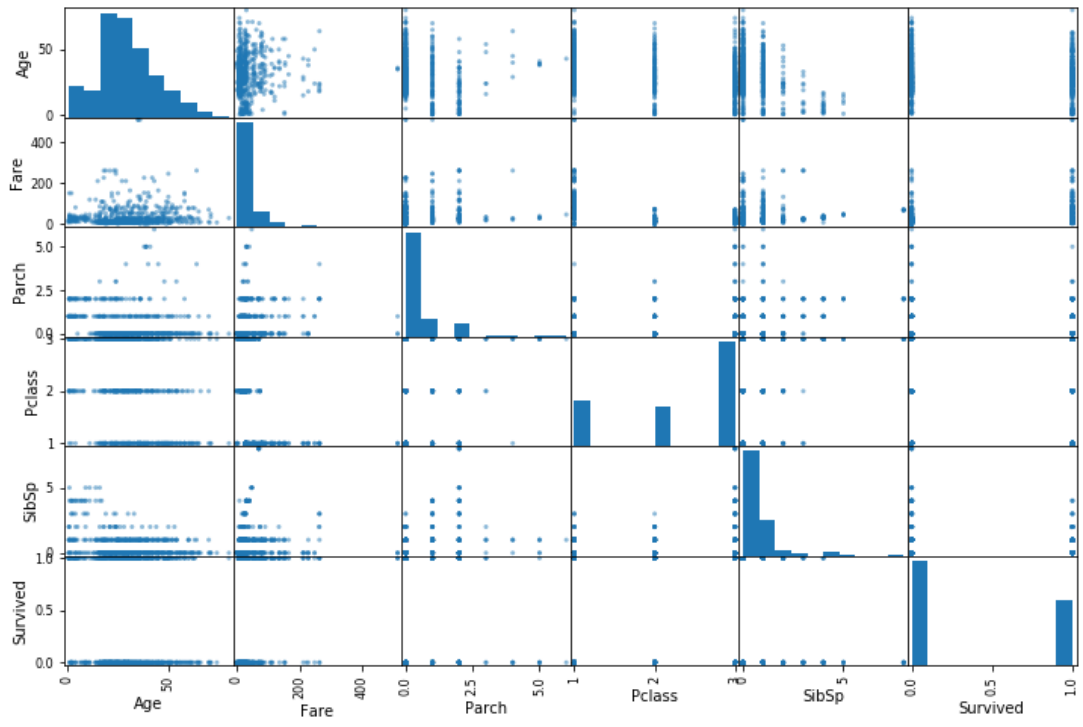
Lab Notebook

2020-04-04 Exploratory Analysis of the Data

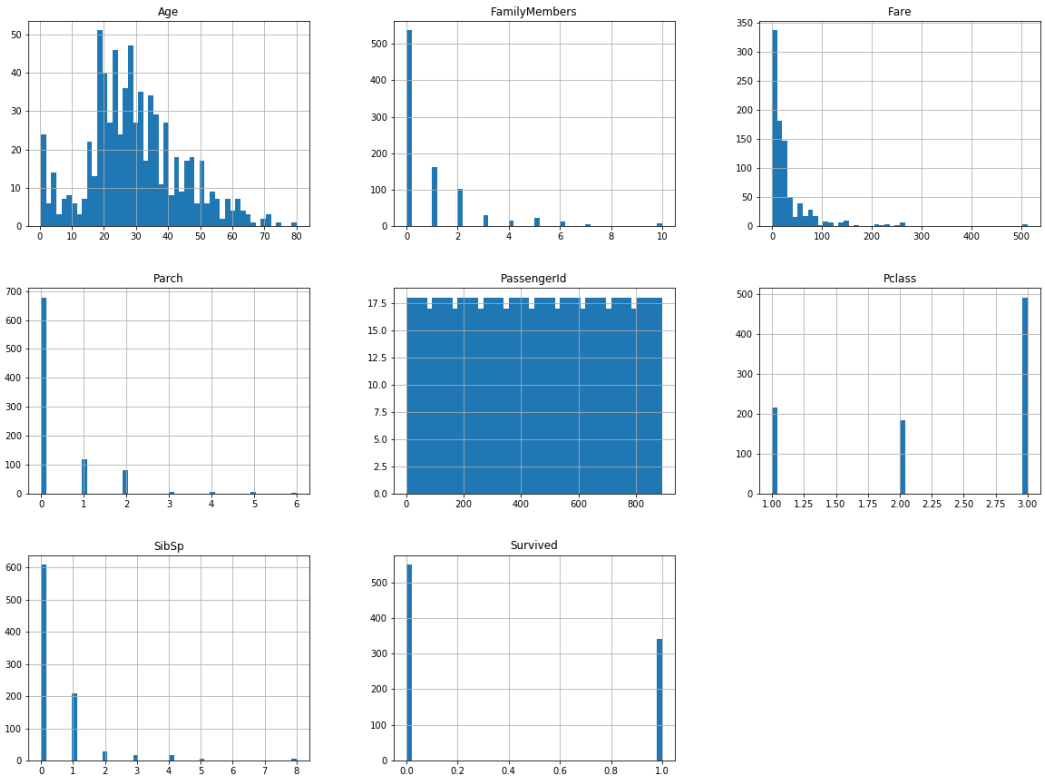
Findings

There are 891 observations. Some data is missing for 'age' and 'cabin'. The overall survival rate is 38%. Mean age of all passengers is 30. The Fare and SibSp variables exhibit significant positive skew. The Fare and Pclass have the highest correlations with the survived variable.

Correlation Matrix



Histogram



Interpretations

- Since Fare and Pclass have the highest correlations with survived, they will likely be the most useful variables in predicting who survives.

Next Steps

- We will need to decide what to do with missing age and cabin data. Since so much of the cabin data is missing, we might need to throw that variable out. For age we can likely start by imputing missing data with the median age.
- Will likely need to convert the Pclass and Survived variables from numeric to categorical variables.
- SibSp, Parch could either be modeled at categorical or numeric data.
- If I were to build a first model, I would likely want to try out a multiple regression model.

2020-04-19

Problem Framing

- I am providing evidence regarding characteristics associated survival to a historian writing a book. This indicates to me that the model does not need to be extremely accurate at predicting survival on an individual basis. Rather it is more important to develop a model that is interpretable, and predicts survival reasonably well.
- The author will use my model to develop a narrative describing the types of people that survived the Titanic and why.
- We already know who survived and who perished. It is more important to discover why individuals survived or perished.
- This is a supervised learning problem; the data is labeled.
- This is an offline model. The model will be trained with batch data.
- Model performance will be measured by the area under the receiver operating characteristic (ROC) curve. This measure of performance does incentivize a more predictive model, but it won't incentivize me to build an explainable one.
- Comparable problems include any other classification-type problems.
- I can likely reuse some pre-processing code from previous projects.
- I do not have access to any human expertise from a historian on this problem.
- If I were to solve this problem manually, I would calculate the average survival rate by characteristic. Then use that to guide the historian's narrative.
- Assumptions:
    - Data quality is a minor to non-existent issue. I'm assuming that all of the survival classifications are correct.

Experiment Plan

I plan on building quick Logistic Regression and Bayes Naive Classifier models to set baseline model predictions. Both the models and the pre-processing pipelines will be very simple. Neither will include any feature engineering.

Logistic Regression Pre-processing and Model Decisions

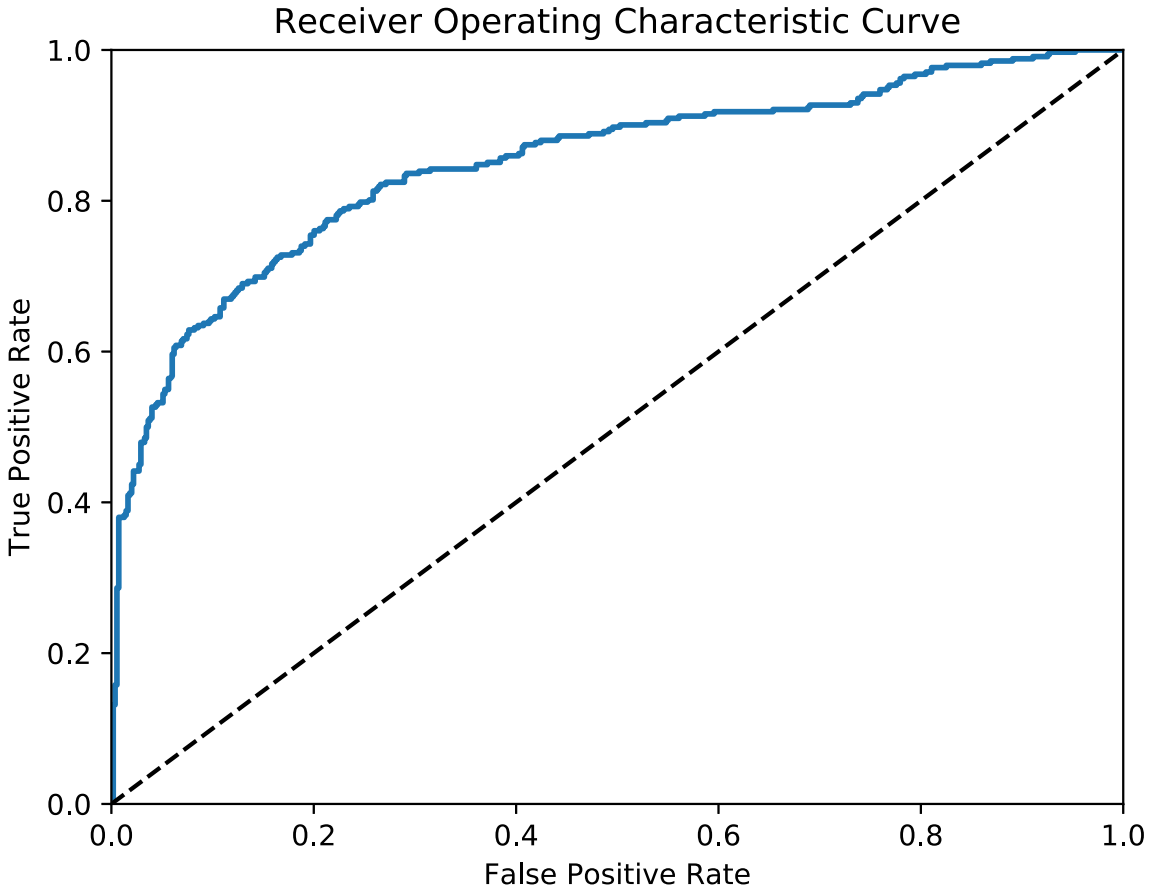
- I decided to throw out the following variables: Name, Ticket, and Cabin. I don't think Name or Ticket will have much predictive value. There is a lot of missing data for the Cabin variable.
- Missing categorical data was imputed with the most frequent values.
- I encoded categorical variables as dummy variables.
- Missing numeric data was imputed with median values.
- The Logistic Regression model parameters were:

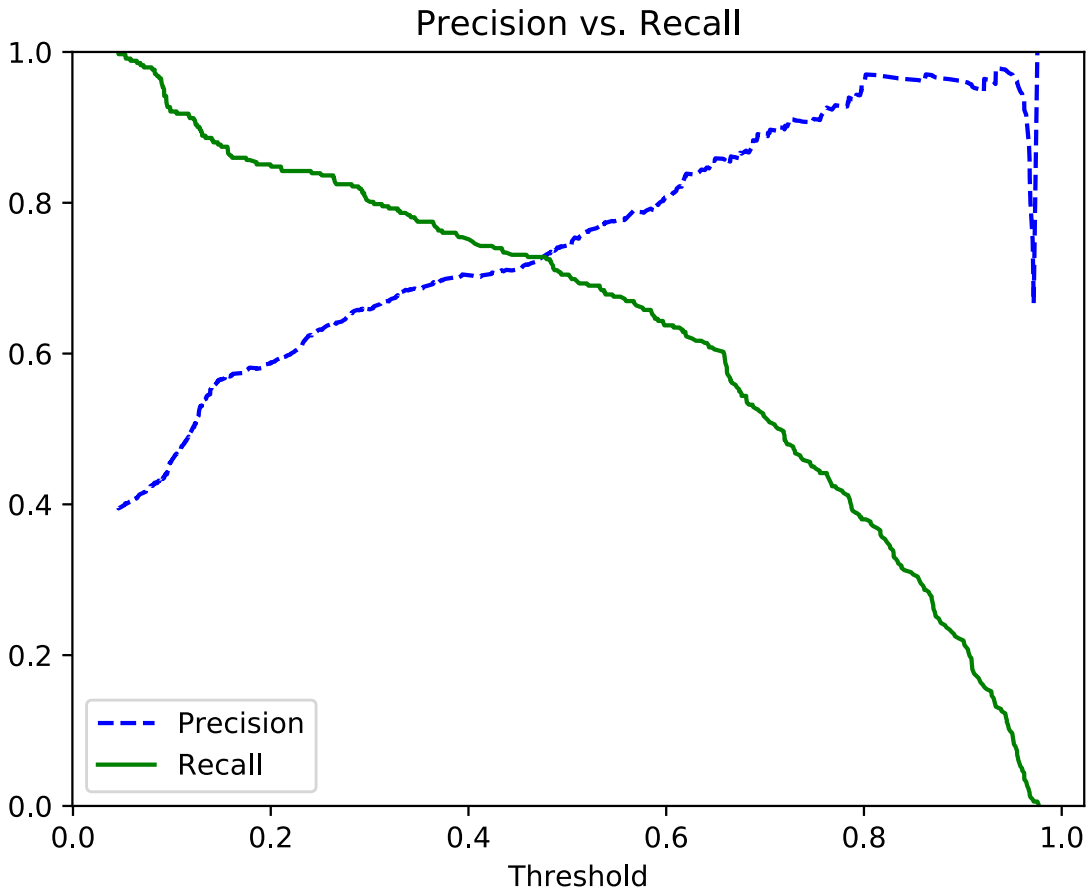
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=94, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

- I set the `penalty` parameter to `"none"`, turning off any regularization.
- I got a convergence warning on the first run, so I upped the `max_iter` parameter to `1000`.
- I used a 10-fold cross validation design with the default scoring technique: "predict_proba". From there I could compute all the necessary metrics.

Logistic Regression Results and Interpretation

Metric	Value
Precision	0.7438271604938271
Recall	0.7046783625730995
Area Under Curve	0.8491675454574507
Accuracy	0.7934904601571269


Receiver Operating Characteristic Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



The baseline Logistic Regression model was able to predict whether an individual would survive or not 79% of the time. The model is a significant improvement over the 38% base rate of survival. However, since this is a very simple model, there may be room for improvement still.

Next Steps

In the next iteration of this model, I would like to experiment with L1 and L2 regularization. That means that I will also need to scale the numeric features in my pre-processing pipeline. Additionally, I would like to vary the amount of regularization using the `C` parameter.

2020-04-22

Experiment Plan

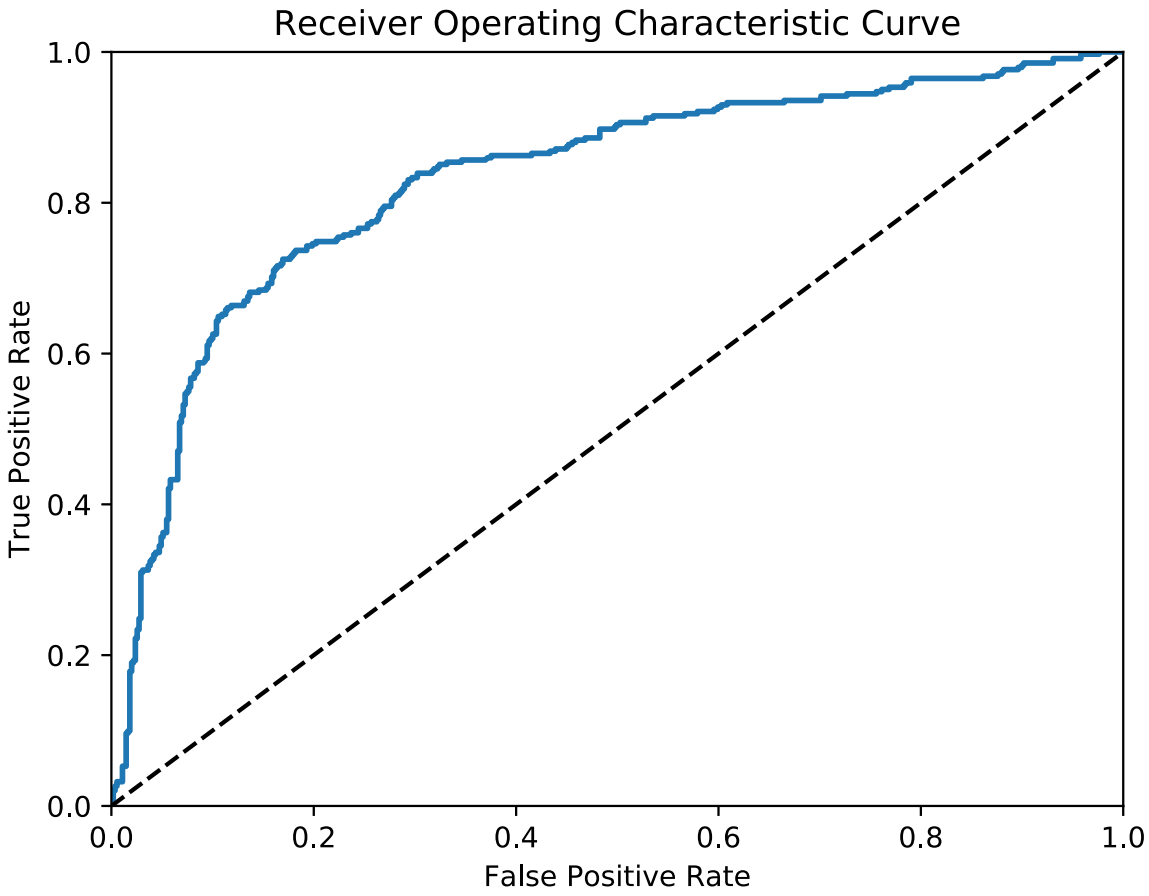
Build a simple baseline Bayes Naive Classifier model. Compare the results to your simple logistic regression model.

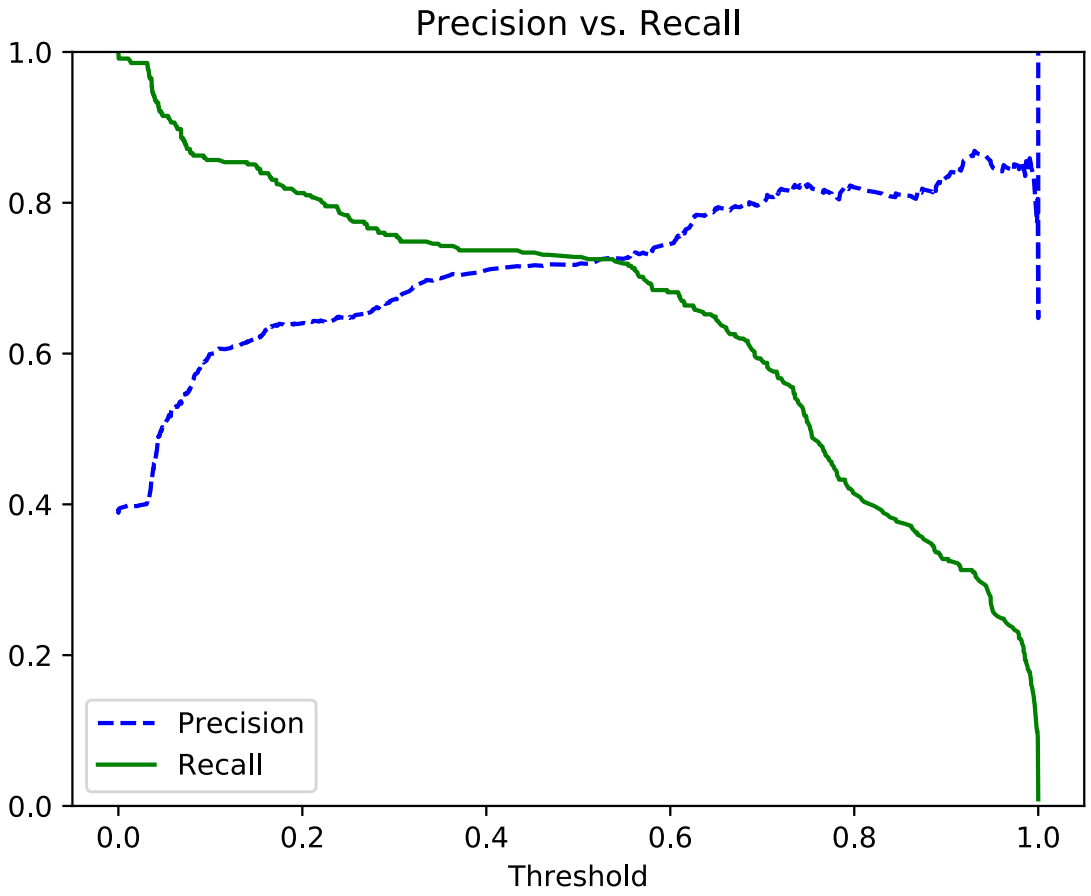
Bayes Naive Classifier Pre-processing and Modeling Decisions

- I decided to throw out the following variables: Name, Ticket, and Cabin. I don't think Name or Ticket will have much predictive value. There is a lot of missing data for the Cabin variable.
- Missing categorical data was imputed with the most frequent values.
- I encoded categorical variables as numbers with an Ordinal Encoder.
- Missing numeric data was imputed with median values.
- I choose to use a GaussianNB model.
- I used a 10-fold cross validation design with the default scoring technique: "predict_proba". From there I could compute all the necessary metrics.

Bayes Naive Classifier Results and Interpretation

Metric	Value
Precision	0.7196531791907514
Recall	0.7280701754385965
Area Under Curve	0.8313733635850404
Accuracy	0.7867564534231201


Receiver Operating Characteristic Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



The simple Naive Bayes model performs remarkably similar to the simple Logistic Regression model.

Submitting this model to Kaggle resulted in a score of 0.75598 (https://www.kaggle.com/brianelinsky) (Account User ID 4810027)

Next Steps

- If I decide to improve on this model, I could use a normal scaler on the features. The Naive Bayes model assumes the data is normally distributed, so this might improve the performance.

2020-04-23

Experiment Plan

I plan on implementing a more complex logistic regression model. I'll experiment with L1 and L2 regularization. In pre-processing, I'll need to scale the numeric features. I'll find the optimal amount of regularization using a grid search.

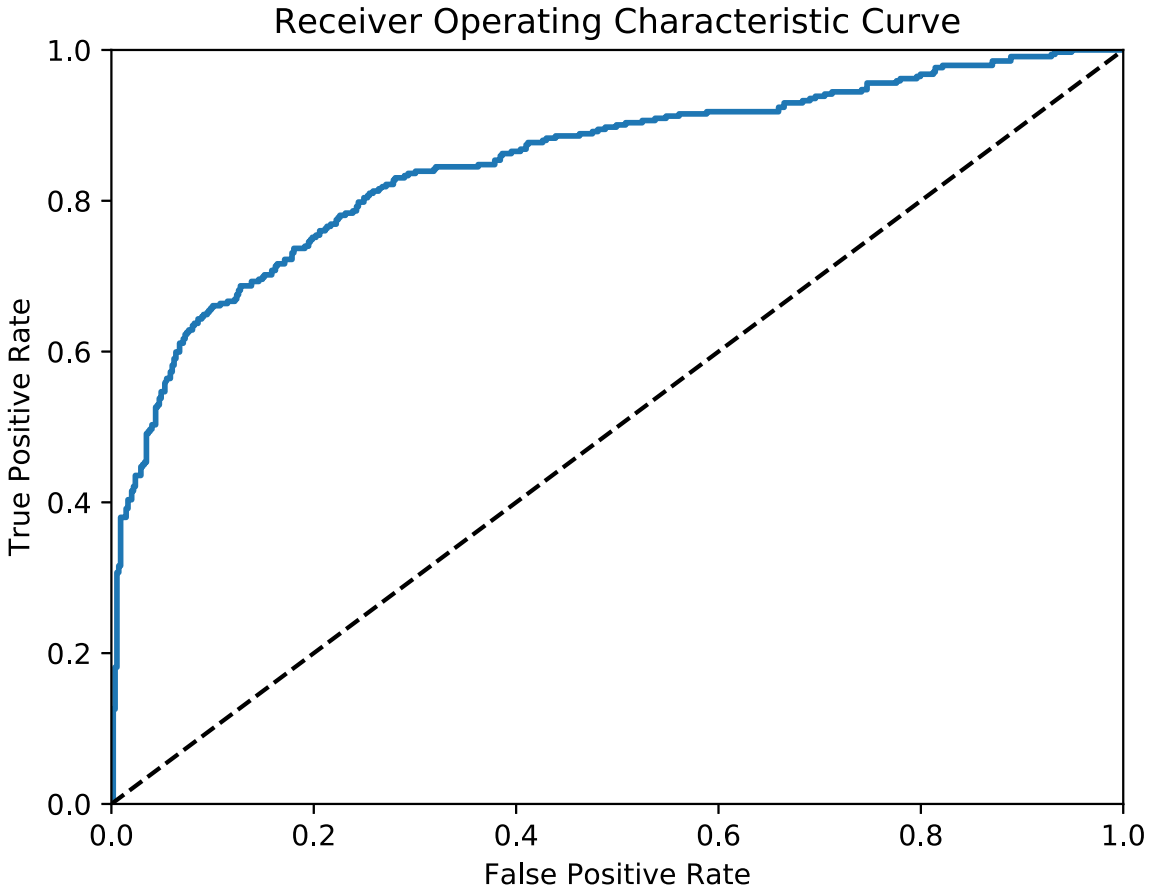
Logistic Regression Pre-processing and Modeling Decisions

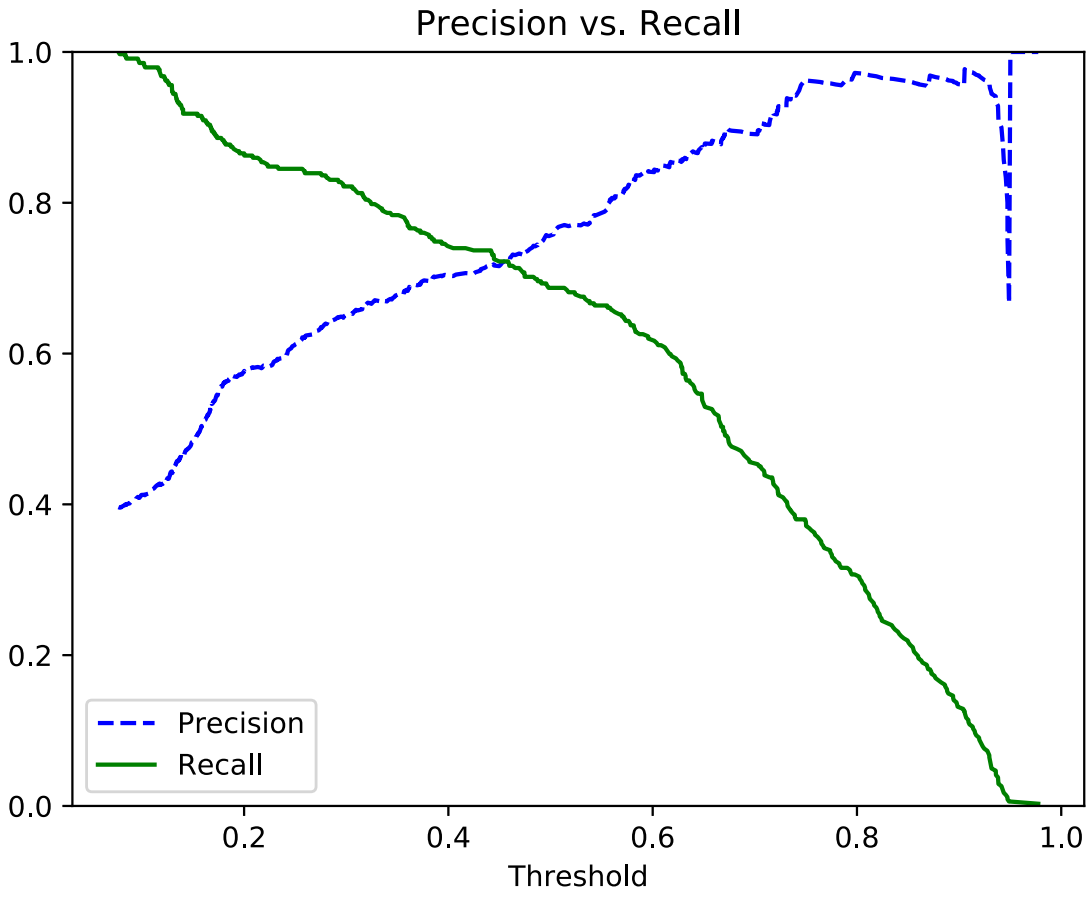
- I retained most of the pipeline from my first logistic regression model.
- I added a RobustScaler to the numeric pipeline.
- I decided to vary the following parameters in a grid search.
- The grid search will use 10-fold cross validation.
- The grid search will use a ROC_AUC scoring method.
- The logistic regression model parameters were:

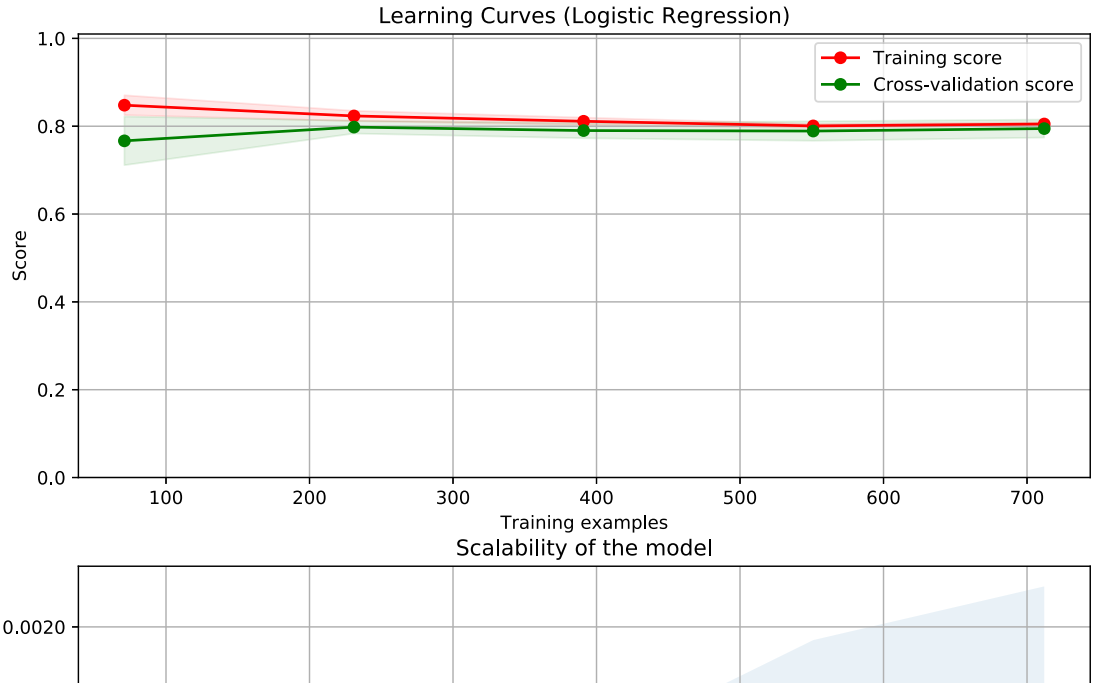
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001,
verbose=0,
                   warm_start=False)
```

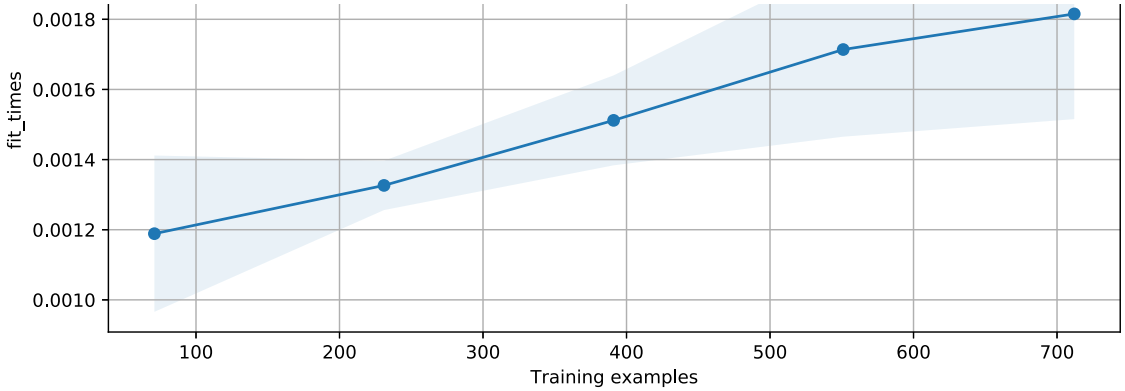
Logistic Regression Results and Interpretation

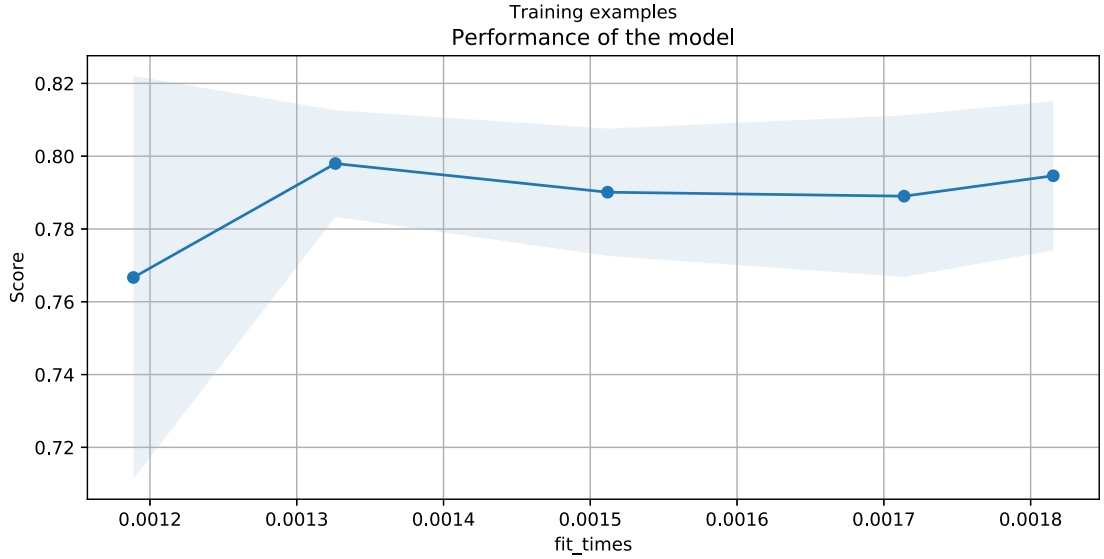
Metric	Value
Precision	0.7580645161290323
Recall	0.6871345029239766
Area Under Curve	0.8313733635850404
Accuracy	0.7867564534231201


Receiver Operating Characteristic Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Precision vs. Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Threshold
Precision
Recall


Learning Curves (Logistic Regression)
Training score
Cross-validation score
Score
1.0
0.8
0.6
0.4
0.2
0.0
100
200
300
400
500
600
700
Training examples
Scalability of the model
0.0020





Adding in regularization did not substantially impact the results. Precision, accuracy, and AUC increased, while recall decreased. Overall the model is slightly better, but with added complexity.

If you look at the learning curve for this model, most of the learning is completed over about the first 230 training examples. After the training score and cross-validation score have converged, additional examples provide no learning benefit. This indicates to me that the model suffers from a bias problem, not a variance problem. Adding more training examples won't make the model better. Rather, I'll need to add complexity to the model to improve it. I could do this by using a different model (that has more degrees of freedom), or by engaging in some feature engineering.

Submitting this model to Kaggle resulted in a score of 0.76555 (https://www.kaggle.com/brianelinsky) (Account User ID 4810027)

Next Steps

This is a pretty popular dataset. From here, I could look at models that others have built, and see what sorts of pre-processing they are employing. I think that most of the improvements to this model will likely be achieved with better feature processing and engineering.