Case Study 2: Lending Club

Team 4: Emily Strong and Raksha Kaverappa

Main Code Repository: <https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/tree/master/Assignment%202>

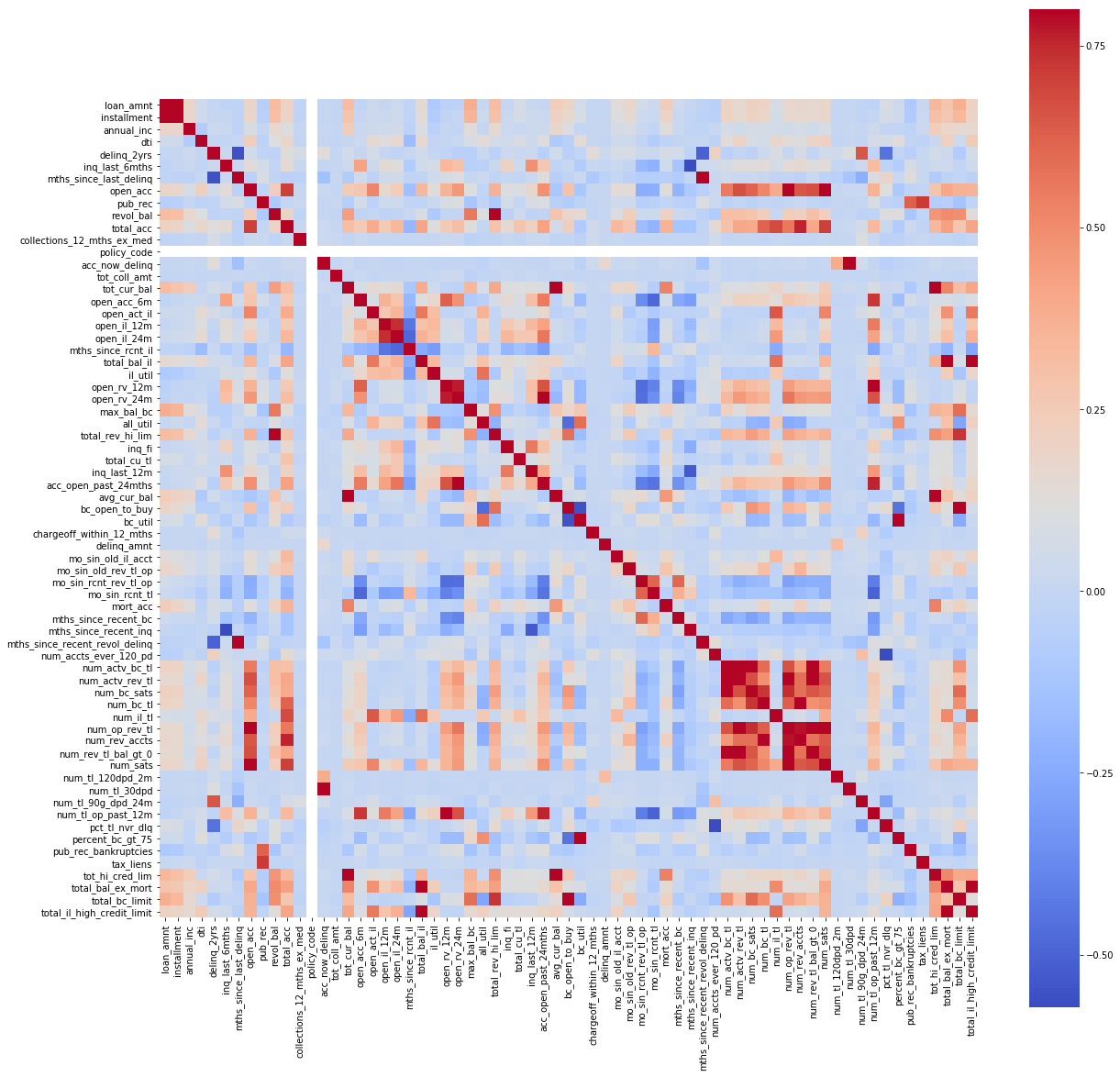
**Part 1A: Data Wrangling**

To scrape the approved loan data:

* We created a session with Lending Club investor credentials.
* We scraped from a hidden div on the page the names of the zip files that need to be accessed, and then read the csvs in each zip file into a data frame.
* We then merged the data frames and added a timestamp and the set name.
* We dropped all columns with >50% missing data.
* We removed columns that are computed by Lending Club after the loan application based on the data dictionary.
* We used a correlation heat map (shown below) and the data dictionary to eliminate redundant columns.
* We did missing value substitutions using mean, median, and max for annual\_inc, mths\_since\_recent\_inq and mths\_since\_last\_delinq.
* We dropped any rows with missing values for the remaining features, cumulatively removing 6% of the data.
* We reformatted numeric features present as strings, and created dummy variables for the categorical features.

To scrape the declined data:

* We followed a similar process but without the authenticated session.
* The only feature with >50% missing data was Risk Score, with 58%. We chose to keep it because of the critical
* For Risk Scores, we converted Vantage scores (applications dated on or after November 6, 2013) to use the same scale as FICO scores.
* We dropped rows with missing location information, and used mean and mode substitutions for the remaining features.
* We dropped rows with a negative DTI, removing 6% of the data.

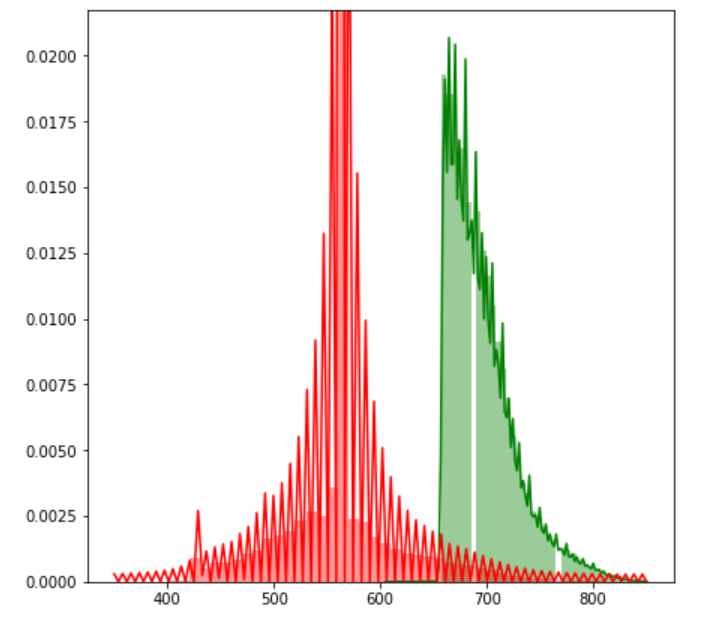
****

**Correlation Heat Map for Loan Features with <=50% missing data**

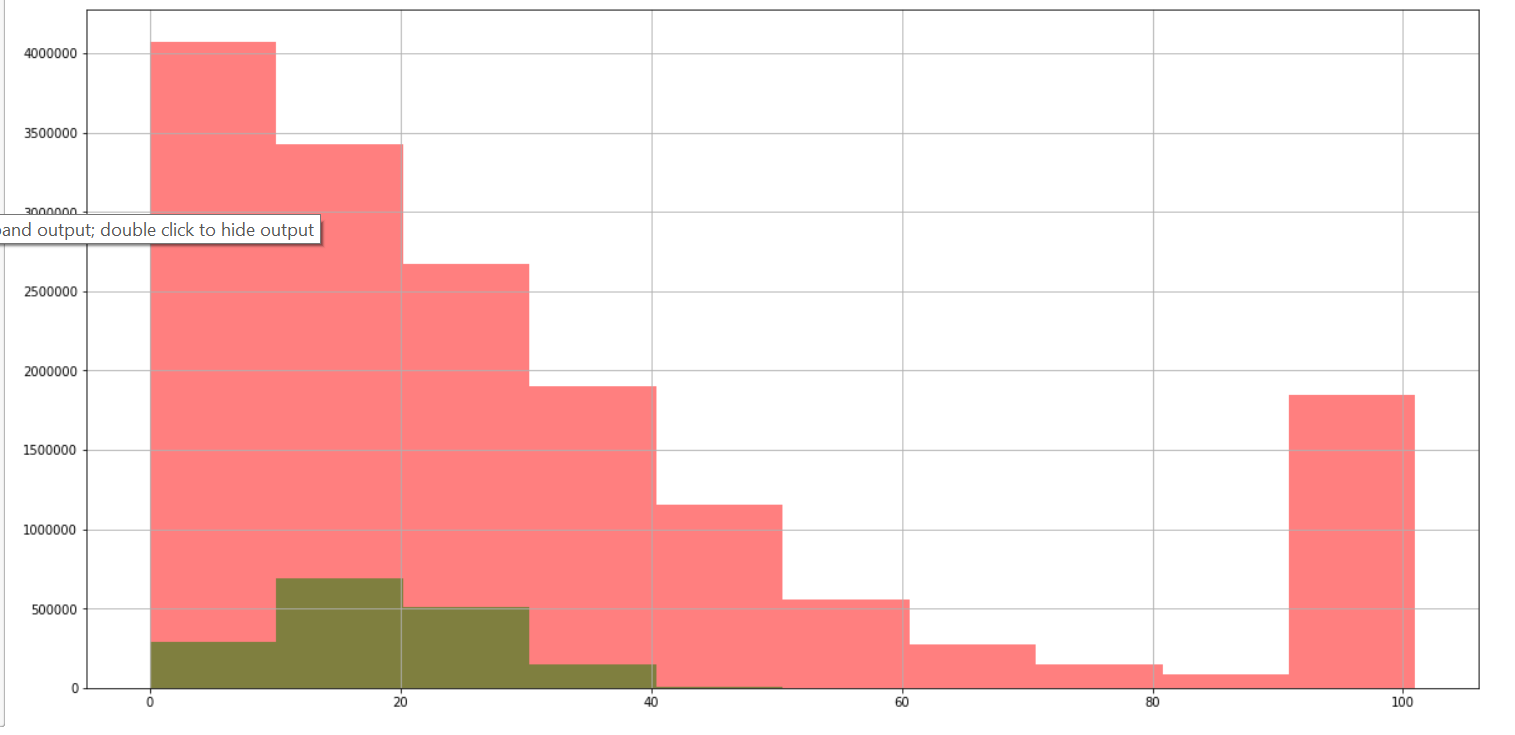
**Part 1B: Exploratory Data Analysis**

*EDA notebook:* <https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/blob/master/Assignment%202/Part%201/Exploratory%20Data%20Analysis.ipynb>

* For the first part of our analysis, we made a comparison study of the FICO score distributions of the accepted and declined datasets. We observed that most of the accepted FICO scores were below 650 approximately. The graph is as shown below:

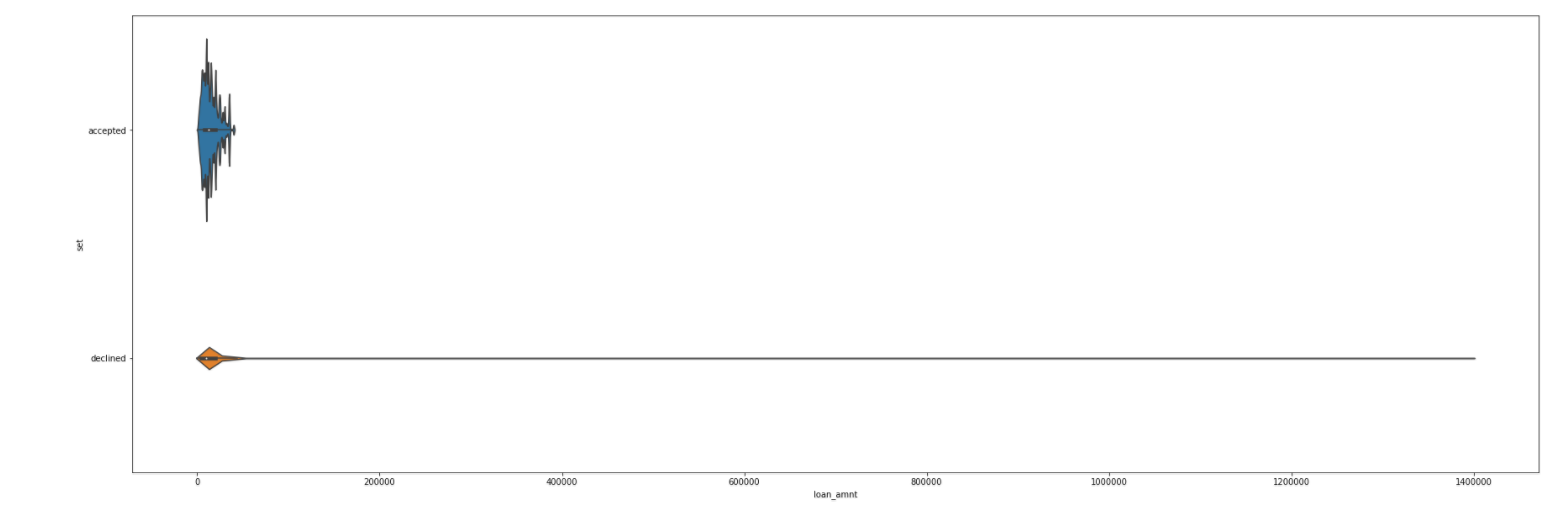


* The Debt to Income ratio distribution for the declined and the approved data is as shown below.



We notice that the declined data has higher DTI when compared to approved data.

* The distribution of loan amounts is as shown below:



We observe that there are several outliers for the declined data.

**Part 2A: Classification**

We combined the approved and declined datasets and used the following features to classify the model:

* Loan amount
* FICO score
* Debt to Income ratio
* Employment length
* Policy code

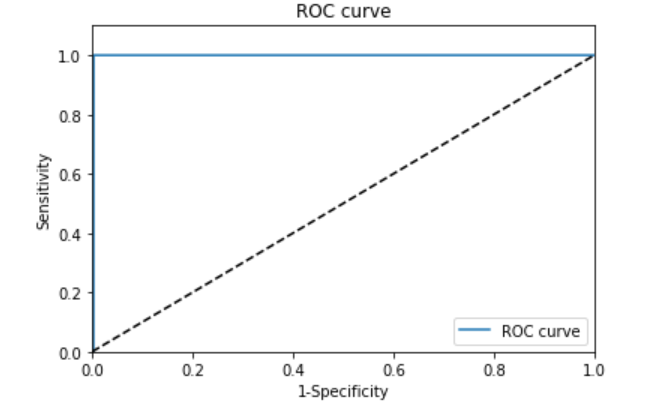
The confusion matrix and ROC curve for all the models shown below. For API deployment we chose to use Random Forest.

**Confusion Matrix:**

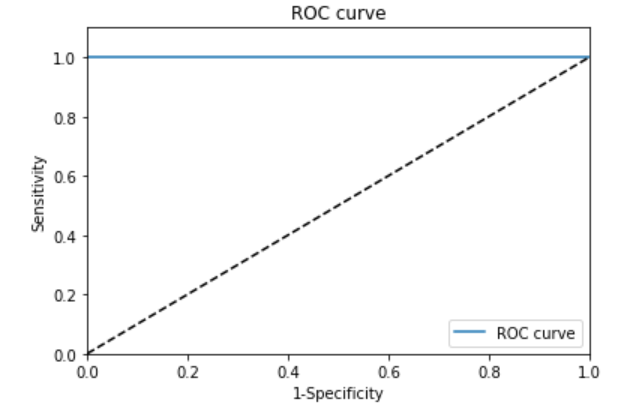
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **True Positives** | **False Positives** | **True Negatives** | **False Negatives** |
| Logistic Regression | 3216939 | 8949 | 302817 | 108 |
| Random Forest | 3226328 | 0 | 302485 | 0 |
| Neural Networks | 3224991 | 0 | 303822 | 0 |

**ROC curves:**

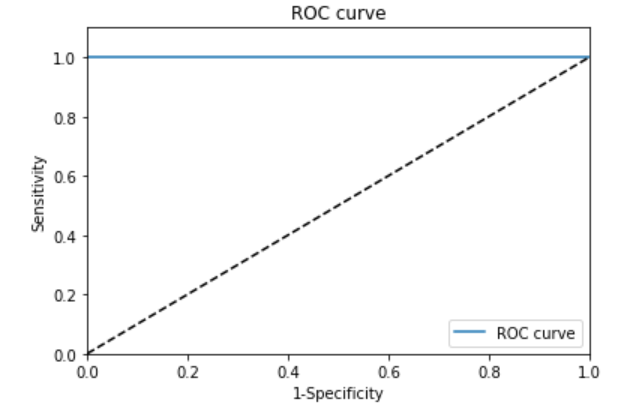
Logistic Regression:



Random Forest:



Neural Networks:



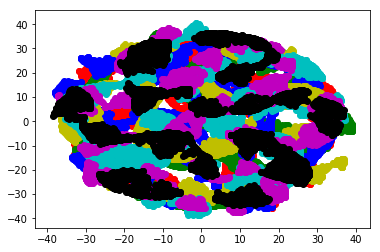
**Part 2B: Clustering**

For selecting KMeans cluster assignments, we compared various iterations against the grade column since these are the actual clusters used by Lending Club. Our best result was 7 clusters with default parameters and all features we are using for prediction. In comparing it to the grades column, four grades have a unique mode cluster, and three share a mode: B, C, and D. Since these are adjacent grades we believe the features being used to group them together are not random.

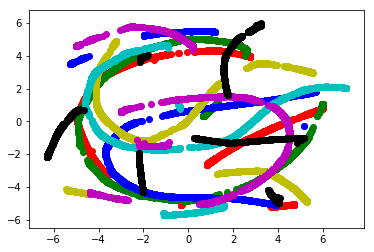
For the manual clusters, we decided to create 6 clusters based on FICO score.

To visualize the data with TSNE we took a random sample of 10% of the data.

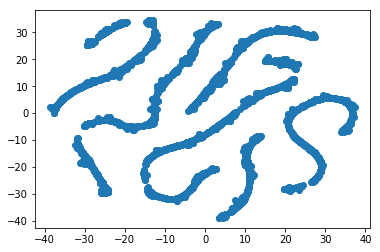
For TSNE, we created two plots of our KMeans clusters, one with a lower perplexity (100) and more steps (500), and one with a higher perplexity (200) but fewer steps (250). For comparison, we also ran one on the no-clusters data.



**TSNE – Perplexity = 100**



**TSNE – Perplexity = 200**



**TSNE – No Clusters**

As the various plots illustrate, there is significant overlap in the KMeans clusters.

**Part 2C: Prediction**

For prediction, we are using the features:

* loan\_amnt
* term
* emp\_length
* annual\_inc
* dti
* delinq\_2yrs
* fico\_range\_low
* mths\_since\_last\_delinq
* open\_acc
* revol\_bal
* revol\_util
* total\_acc
* collections\_12\_mths\_ex\_med
* policy\_code
* application\_type
* tot\_cur\_bal
* bc\_util
* chargeoff\_within\_12\_mths
* mo\_sin\_old\_il\_acct
* mo\_sin\_rcnt\_tl
* mort\_acc
* mths\_since\_recent\_bc
* mths\_since\_recent\_inq
* num\_accts\_ever\_120\_pd
* num\_actv\_bc\_tl
* num\_actv\_rev\_tl
* num\_bc\_tl
* num\_il\_tl
* num\_rev\_accts
* num\_rev\_tl\_bal\_gt\_0
* num\_sats
* num\_tl\_30dpd
* num\_tl\_90g\_dpd\_24m
* num\_tl\_op\_past\_12m
* percent\_bc\_gt\_75
* pub\_rec\_bankruptcies
* tax\_liens
* total\_bc\_limit
* home\_ownership
* addr\_state

These features were selected during our data cleansing. When we used various methods of feature selection while creating prediction models we found that removing any of these features increased the error rate for the models. Regarding parallelizing the code, we were not able to understand how to implement the multiprocessing Python library and in reading up on it we learned that it does not work well in Jupyter Notebook. We thus decided to parallelize our code by creating separate notebooks for no clusters, KMeans clusters and manual clusters for Neural Networks, Random Forests and KNN so that the notebooks could be run concurrently.

The full error results (MAE, RMS, and MAPE) for our models are in 'Prediction Results.xlsx'. Below are samples of the results for two of our clusters. With our total data set we experimented with the parameters, but for the clustered data we ran each with only the parameters we selected as optimal from working with the no-clusters data. For Neural Network we used hidden layers of 50, 20, 10, for Random Forest we used 60 decision trees, and for KNN we used 5 neighbors.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **KMeans Cluster 0** | Jupyter Notebook | | | | | | Azure ML | |
| Metric | Train RMS | Test RMS | Train MAE | Test MAE | Train MAPE | Test MAPE | Test MAE | Test RMS |
| Linear Regression | 3.57 | 3.53 | 2.77 | 2.76 | 24.27 | 24.40 | N/a | N/a |
| Neural Network | 3.07 | 3.20 | 2.34 | 2.44 | 19.95 | 20.79 | 2.59 | 3.32 |
| Random Forest | 1.23 | 3.23 | 0.93 | 2.48 | 7.99 | 21.37 | 2.56 | 3.34 |
| KNN | 3.80 | 4.63 | 2.96 | 3.62 | 26.61 | 32.67 | N/a | N/a |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **KMeans Cluster 5** | Jupyter Notebook | | | | | | Azure ML | |
| Metric | Train RMS | Test RMS | Train MAE | Test MAE | Train MAPE | Test MAPE | Test MAE | Test RMS |
| Linear Regression | 2.85 | 2.84 | 2.24 | 2.23 | 17.10 | 17.04 | N/a | N/a |
| Neural Network | 2.47 | 2.55 | 1.91 | 1.96 | 14.06 | 14.48 | N/a | N/a |
| Random Forest | 0.97 | 2.57 | 0.74 | 1.99 | 5.58 | 14.79 | 2.06 | 2.65 |
| KNN | 3.37 | 4.11 | 2.68 | 3.27 | 20.96 | 25.71 | N/a | N/a |

For our no-cluster data and KNN clusters, Neural Network gave the best results in Jupyter Notebook by a small margin, and Random Forest gave the best results for our manual clusters, despite overfitting the training data. However, when deploying to Azure ML we discovered that it only allows one layer for Neural Networks, and in comparing Neural Networks to Random Forests we found that Random Forests performs marginally better on that platform (shown for KMeans cluster 0 above). Since the difference between the two was marginal in both environments, we decided to use Random Forests for all prediction APIs.

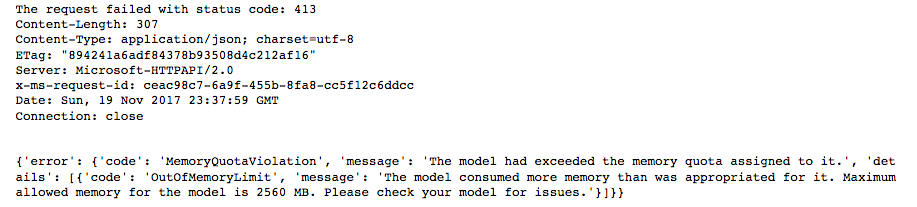
An example Azure implementation:

**Part 2D: Deployment**

We deployed our final models in Jupyter Notebook. To use it, it can be run with a demo loan applicant or you can input the values manually to calculate for a new user. Features that anyone using the script, whether a loan officer or a potential applicant, would know are required and features obtained from a credit report are optional. If an optional feature is left blank, a default value is assigned. The script then converts the state and home ownership status to one-hot values and calls the accept/decline categorization API. If the application is declined it prints a message saying so, otherwise it assigned the applicant manual and KMeans clusters and calls the interest rate prediction APIs. We compare the returned interest rates and print out the largest one:

../../../Screen%20Shot%202017-11-19%20at%208.37.19%20PM.png

With our "no cluster" API we receive an error message that the model is larger than Azure's memory limit:



We researched the error and the workaround is to use Azure's batch processing, which requires uploading the input to Azure manually to convert it into a blob format. We excluded this API from the workflow and include it in a separate cell at the bottom of the notebook to display the error we are getting.