Final Project Report

By Robert Syvarth & William Headrick | RS42999 & WH5965

1. How did your team go about tackling this problem?
We started out by reading through a lot of the advice on the Kaggle forums. After reading through and getting some ideas we determined that it would be most beneficial if we wrote our solution without using any existing code as a template. This way we would have to understand everything that we were doing – not just blindly copying it from elsewhere on the Internet.

We went with an iterative approach starting out with a very simple solution that just involved a single classifier to prove that our programming was correct and over time adding on additional processing steps in order to improve upon this baseline.

2. Which methods/algorithms did you try?
We ended up trying a lot of different classifiers from Scikit early on. We tried the Bernoulli Naive Bayes classifier, LogisticRegression, AdaBoost, GradientBoost, SVC, and RandomForest. For each of these classifiers we did around a hundred test trains with varying parameters in order to evaluate the efficacy of that particular classifier on our data. We also determined that we needed a method for combining our predicted results so we researched various options there and experimented with simple linear combinations which were simple and performed pretty well. Finally we investigated various forms of feature combination and selection. Many of the existing solutions used feature combination in order to expand the dimensionality of the problem followed by greedy feature selection. In addition to these options we also looked into manual, heuristic-based, combination and Recursive Feature Elimination.

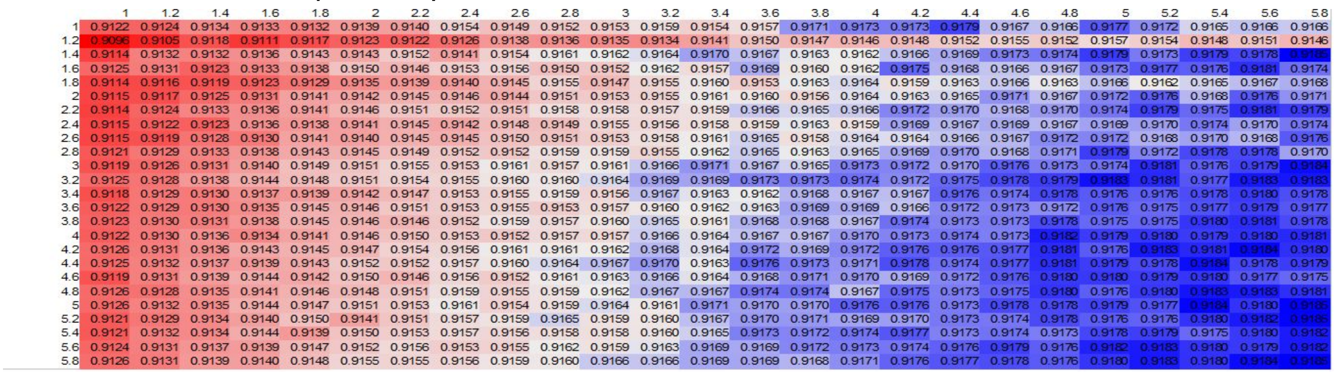
3. What is your final methodology? Walk through it in detail, starting from data pre-processing. Explain all the machine learning algorithms(s) you used as well as the parameters you chose. Also discuss any external tools or libraries that you used.
Our final solution uses a combination of three different classifiers as well as feature combination and Recursive Feature Elimination in order to predict employee access requirements.

First we use Numpy's loadtxt function in order to read CSV data from disk. This gives us easy access to the data without having to worry about details of CSV parsing. The first step we take once we have all of the data loaded is to expand the dimensionality of the data. In order to achieve this we consider all pairs of features in the original data as well as the original features themselves. This allows us to coax a little more subtlety out of some of our classifiers by selecting specific sets of combined features.

The next step is feature extraction. Once we have expanded our set of features we need to remove those which are redundant or otherwise unhelpful to our classifiers. In order to do this we use Scikit's Recursive Feature Elimination class. This allows us to pass a classifier and a dataset it returns a list of the most relevant N features in that classifier, in our case we ended up selecting 30 features in total. We came to this result through simply running our cross validation with different numbers of features until we reached a maximum. We used the LogisticRegression classifier for this step since, as we will discuss later, it accounts for the greatest weight in our final model. Another thing worth noting is that we scale all of our features from 0-1 prior to passing them to the feature elimination step. We include this step because otherwise features with larger absolute values are preferred. The alternative would be to attempt a one-hot encoding at this step but the number of features produced by a one-hot encoding prevent any reasonable feature selection from occurring after it has occurred.

After we have selected our features we simply remove those not in the top 30 from our original datasets. The next step is to perform a one-hot encoding on this newly reduced dataset so that it is ready for our classifiers. Once the one-hot encoding is complete we are ready to start training our classifiers. In the end we decided to use three models for the machine learning step of our project: LogisticRegression, RandomForest, and SVC. These were chosen from the list of classifiers which we evaluated due to their performance on our cross validation tests.

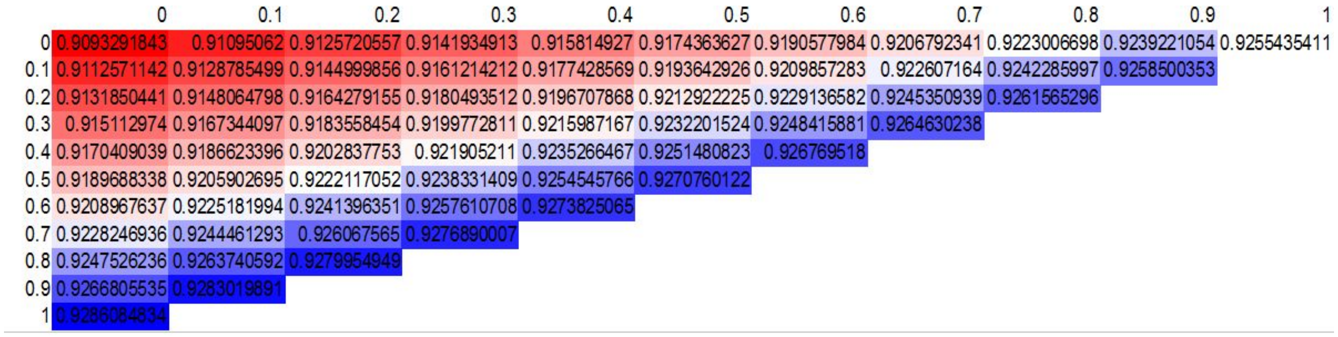
While choosing the best parameters for these models it became pretty clear that there were only a few which actually significantly impacted our results. The most important for both the LogisticRegression model and SVC is C. The C value essentially defines how much regularization is applied to the model in order to prevent overfitting. As you can see in **Figure 1** there were some pretty clear trends in our cross validation results based on the value of C both these classifiers. We ended up choosing values of 4.4 and 1.4 for these two models respectively based on these tests.



	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6	3.8	4	4.2	4.4	4.6	4.8	5	5.2	5.4	5.6	5.8
1	0.9122	0.9124	0.9134	0.9133	0.9132	0.9139	0.9140	0.9154	0.9149	0.9152	0.9153	0.9159	0.9154	0.9157	0.9171	0.9173	0.9173	0.9179	0.9167	0.9166	0.9177	0.9172	0.9165	0.9166	0.9166
1.2	0.9096	0.9105	0.9118	0.9111	0.9117	0.9123	0.9122	0.9126	0.9138	0.9136	0.9135	0.9134	0.9141	0.9150	0.9147	0.9146	0.9148	0.9152	0.9155	0.9152	0.9157	0.9154	0.9148	0.9151	0.9146
1.4	0.9114	0.9132	0.9132	0.9136	0.9143	0.9143	0.9152	0.9141	0.9154	0.9161	0.9162	0.9164	0.9170	0.9167	0.9163	0.9162	0.9166	0.9169	0.9173	0.9174	0.9179	0.9173	0.9179	0.9178	0.9185
1.6	0.9125	0.9131	0.9123	0.9133	0.9138	0.9150	0.9146	0.9153	0.9156	0.9150	0.9152	0.9162	0.9157	0.9169	0.9160	0.9162	0.9175	0.9168	0.9166	0.9167	0.9173	0.9177	0.9176	0.9181	0.9174
1.8	0.9114	0.9116	0.9119	0.9123	0.9129	0.9135	0.9139	0.9140	0.9145	0.9155	0.9147	0.9155	0.9160	0.9153	0.9163	0.9164	0.9159	0.9163	0.9166	0.9163	0.9166	0.9162	0.9165	0.9167	0.9168
2	0.9115	0.9117	0.9125	0.9131	0.9141	0.9142	0.9145	0.9146	0.9144	0.9151	0.9153	0.9155	0.9161	0.9160	0.9156	0.9164	0.9163	0.9165	0.9171	0.9167	0.9172	0.9176	0.9168	0.9176	0.9171
2.2	0.9114	0.9124	0.9133	0.9136	0.9141	0.9146	0.9151	0.9152	0.9151	0.9158	0.9158	0.9157	0.9159	0.9166	0.9165	0.9166	0.9172	0.9170	0.9168	0.9170	0.9174	0.9179	0.9175	0.9181	0.9179
2.4	0.9115	0.9122	0.9123	0.9136	0.9138	0.9141	0.9145	0.9142	0.9148	0.9149	0.9155	0.9156	0.9158	0.9159	0.9163	0.9159	0.9169	0.9167	0.9169	0.9167	0.9169	0.9170	0.9174	0.9170	0.9174
2.6	0.9115	0.9119	0.9128	0.9130	0.9141	0.9140	0.9145	0.9145	0.9150	0.9151	0.9153	0.9158	0.9161	0.9165	0.9158	0.9164	0.9164	0.9166	0.9167	0.9172	0.9172	0.9168	0.9170	0.9168	0.9176
2.8	0.9121	0.9129	0.9133	0.9138	0.9143	0.9145	0.9149	0.9152	0.9152	0.9159	0.9159	0.9155	0.9162	0.9165	0.9163	0.9165	0.9169	0.9170	0.9168	0.9171	0.9179	0.9172	0.9178	0.9178	0.9170
3	0.9119	0.9126	0.9131	0.9140	0.9149	0.9151	0.9155	0.9153	0.9161	0.9157	0.9161	0.9166	0.9171	0.9167	0.9165	0.9173	0.9172	0.9170	0.9176	0.9173	0.9174	0.9181	0.9176	0.9179	0.9184
3.2	0.9125	0.9128	0.9138	0.9144	0.9148	0.9151	0.9154	0.9155	0.9160	0.9160	0.9164	0.9169	0.9169	0.9173	0.9173	0.9174	0.9172	0.9175	0.9178	0.9179	0.9183	0.9181	0.9177	0.9183	0.9183
3.4	0.9118	0.9129	0.9130	0.9137	0.9139	0.9142	0.9147	0.9153	0.9155	0.9159	0.9156	0.9167	0.9163	0.9162	0.9168	0.9167	0.9167	0.9176	0.9174	0.9178	0.9176	0.9176	0.9178	0.9180	0.9178
3.6	0.9122	0.9129	0.9130	0.9135	0.9145	0.9146	0.9151	0.9153	0.9155	0.9153	0.9157	0.9160	0.9162	0.9163	0.9169	0.9169	0.9166	0.9172	0.9173	0.9172	0.9176	0.9175	0.9177	0.9179	0.9177
3.8	0.9123	0.9130	0.9131	0.9138	0.9145	0.9146	0.9146	0.9152	0.9159	0.9157	0.9160	0.9165	0.9161	0.9168	0.9168	0.9167	0.9174	0.9173	0.9173	0.9178	0.9175	0.9175	0.9180	0.9181	0.9178
4	0.9122	0.9130	0.9136	0.9134	0.9141	0.9146	0.9150	0.9153	0.9152	0.9157	0.9157	0.9166	0.9164	0.9167	0.9167	0.9170	0.9173	0.9174	0.9173	0.9182	0.9179	0.9180	0.9179	0.9180	0.9181
4.2	0.9126	0.9131	0.9136	0.9143	0.9145	0.9147	0.9154	0.9156	0.9161	0.9161	0.9162	0.9168	0.9164	0.9172	0.9169	0.9172	0.9176	0.9176	0.9177	0.9181	0.9176	0.9183	0.9181	0.9184	0.9180
4.4	0.9125	0.9132	0.9137	0.9139	0.9143	0.9152	0.9152	0.9157	0.9160	0.9164	0.9167	0.9170	0.9163	0.9176	0.9173	0.9171	0.9178	0.9174	0.9177	0.9181	0.9179	0.9178	0.9184	0.9178	0.9179
4.6	0.9119	0.9131	0.9139	0.9144	0.9142	0.9150	0.9146	0.9156	0.9152	0.9161	0.9163	0.9166	0.9164	0.9168	0.9171	0.9170	0.9169	0.9172	0.9176	0.9180	0.9180	0.9179	0.9180	0.9177	0.9175
4.8	0.9126	0.9128	0.9135	0.9141	0.9146	0.9148	0.9151	0.9159	0.9155	0.9159	0.9162	0.9167	0.9167	0.9174	0.9174	0.9167	0.9175	0.9173	0.9175	0.9180	0.9176	0.9180	0.9183	0.9183	0.9181
5	0.9126	0.9132	0.9135	0.9144	0.9147	0.9151	0.9153	0.9161	0.9154	0.9159	0.9164	0.9161	0.9171	0.9170	0.9170	0.9176	0.9176	0.9173	0.9178	0.9178	0.9179	0.9177	0.9184	0.9180	0.9185
5.2	0.9121	0.9129	0.9134	0.9140	0.9150	0.9141	0.9151	0.9157	0.9159	0.9165	0.9159	0.9160	0.9167	0.9170	0.9171	0.9169	0.9170	0.9173	0.9174	0.9178	0.9176	0.9176	0.9180	0.9182	0.9185
5.4	0.9121	0.9132	0.9134	0.9144	0.9139	0.9150	0.9153	0.9157	0.9156	0.9158	0.9158	0.9160	0.9165	0.9173	0.9172	0.9174	0.9177	0.9173	0.9174	0.9173	0.9178	0.9179	0.9175	0.9180	0.9182
5.6	0.9124	0.9131	0.9137	0.9139	0.9147	0.9152	0.9156	0.9153	0.9155	0.9162	0.9159	0.9163	0.9169	0.9169	0.9172	0.9173	0.9174	0.9176	0.9179	0.9176	0.9182	0.9183	0.9180	0.9179	0.9182
5.8	0.9126	0.9131	0.9139	0.9140	0.9148	0.9155	0.9155	0.9156	0.9159	0.9160	0.9166	0.9166	0.9169	0.9169	0.9168	0.9171	0.9176	0.9177	0.9178	0.9176	0.9180	0.9183	0.9180	0.9184	0.9185

Figure 1. Heatmap of cross validation scores by Logistic Regression and SVC C values

After we had selected and tuned each of our classifiers we had to determine how to combine their predictions into a single set of values. Our solution was a simple weighted linear combination of our result values. We just needed to determine the weights. In order to do this we took a similar approach to parameter selection and simply tested out many different combinations in order to find a near optimal value, as you can see in **Figure 2**. In this case though we found that despite our cross validation we ended up with an overfit result, so we performed some manual adjustments and ended up using weights of 0.83 for LR, 0.12 for RF, and 0.05 of SVC.



	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.9093291843	0.91095062	0.9125720557	0.9141934913	0.915814927	0.9174363627	0.9190577984	0.9206792341	0.9223006698	0.9239221054	0.9255435411
0.1	0.9112571142	0.9128785499	0.9144999856	0.9161214212	0.9177428569	0.9193642926	0.9209857283	0.922607164	0.9242285997	0.9258500353	
0.2	0.9131850441	0.9148064798	0.9164279155	0.9180493512	0.9196707868	0.9212922225	0.9229136582	0.9245350939	0.9261565296		
0.3	0.915112974	0.9167344097	0.9183558454	0.9199772811	0.9215987167	0.9232201524	0.9248415881	0.9264630238			
0.4	0.9170409039	0.9186623396	0.9202837753	0.921905211	0.9235266467	0.9251480823	0.926769518				
0.5	0.9189688338	0.9205902695	0.9222117052	0.9238331409	0.9254545766	0.9270760122					
0.6	0.9208967637	0.9225181994	0.9241396351	0.9257610708	0.9273825065						
0.7	0.9228246936	0.9244461293	0.926067565	0.9276890007							
0.8	0.9247526236	0.9263740592	0.9279954949								
0.9	0.9266805535	0.9283019891									
1	0.9286084834										

Figure 2. Heatmap of cross validations scores based on weights of individual models

The final step of our classifier is to then combine the predictions from each of our models using the weighted linear combination explained above. We then simply write out our final averaged results to a csv so that we can submit for grading on Kaggle. Using this approach we achieved a private score of 0.89177.

More detailed versions of the data provided here can be found in the Data Analysis.ods file