Progress Report

The following is an update on the progress of Obidroid in terms of **development**

The code for this report is available in an IPython notebook form at:

- [Obidroid Univariate Exploration]
- [Obidroid Multivariate Exploration]
- [Obidroid Unsupervised Learning]
- [Obidroid Classifier Analysis]

— Feedbacks —

- Looking at the cluster analysis, the question is:
    - what is special about the apps in cluster 1 and 7?
    - You can look at those and then do something else with the other apps the are lumped together in 2-6. This might be worth digging into more.
- Here is a way to better assess your classification algorithm. Split it into 50/50 positive and negative tests.
    - Based on the clustering results, choose some of the harder to discriminate items to be the positive items you compare against. Then see how well the classification algorithm works on the 50-50 split."

— Data Preparation —

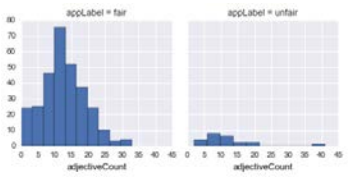
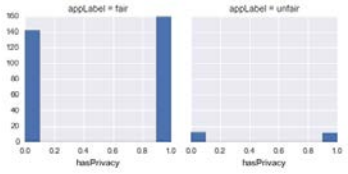
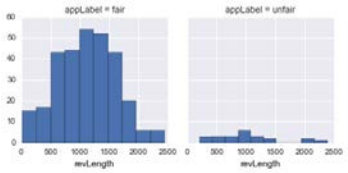
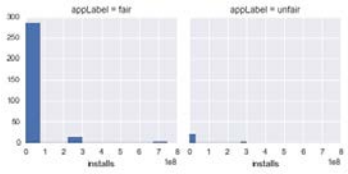
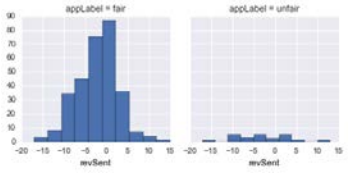
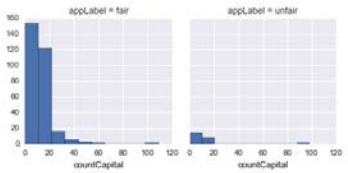
1. The data was prepared from previously extracted `exports/appFeatures.csv` using our crawler and scraper scripts.
2. The data was then **casted to appropriate datatypes** explicitly to avoid any mistakes
3. All variables in the data (i.e. the features) were then normalized using `min-max` **normalization**.

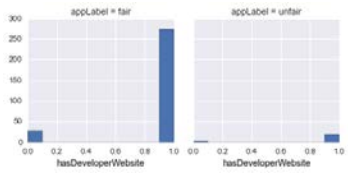
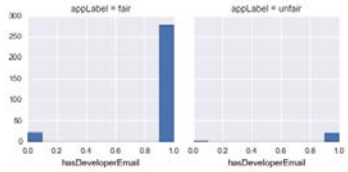
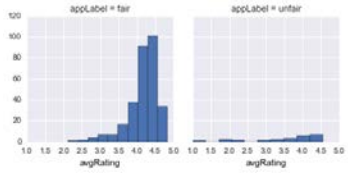
For detailed feature descriptions, please refer Obidroid Project Report

— Univariate Exploration —

Following is an exploration of each variable taken one-by-one at a time.

Here we explore the distributions of each data/feature

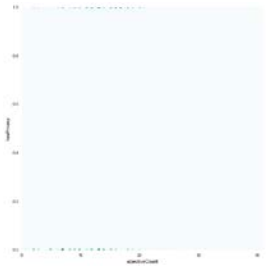
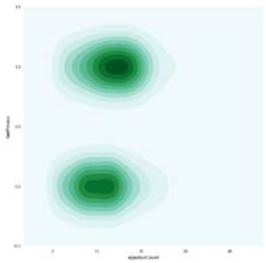
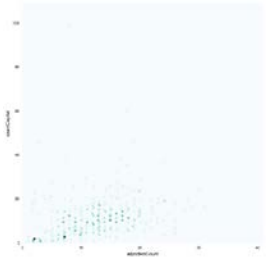
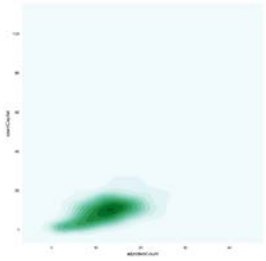
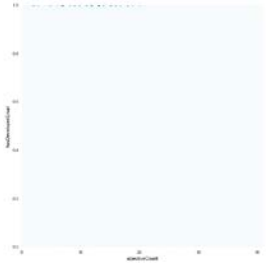
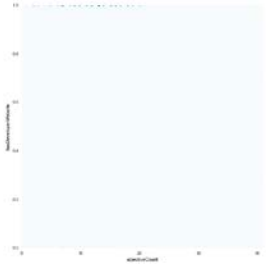
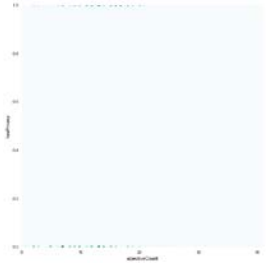
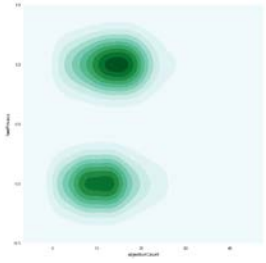
FEATURE	DESCRIPTION	HIST (FAIR/UNFAIR)
`adjectiveCount`	count of all adjectives	
`hasPrivacy`	Does it have a valid privacy url	
`revLength`	Total characters in a review	
`installs`	Total installs of an app	
`revSent`	Aggregate review sentiment	
`countCapital`	Count capital characters in a sentence	
`hasDeveloperWebsite`	Developer website	

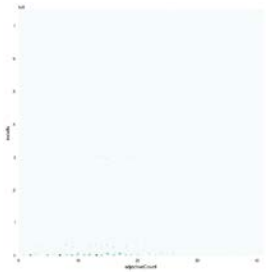
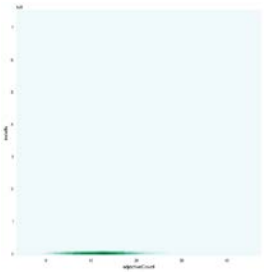
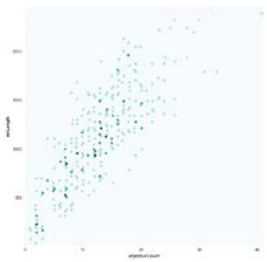
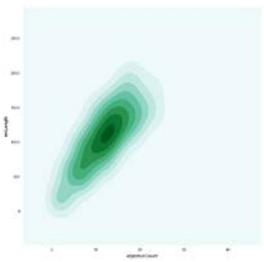
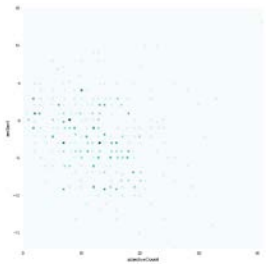
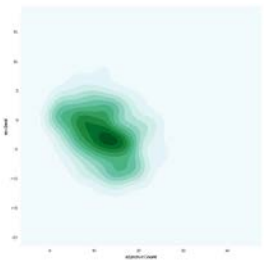
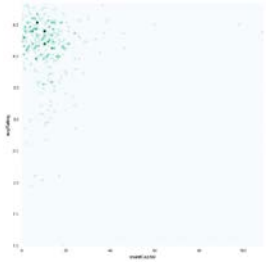
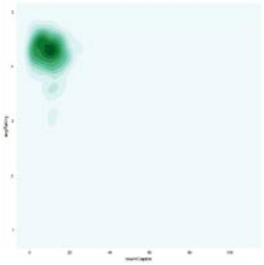
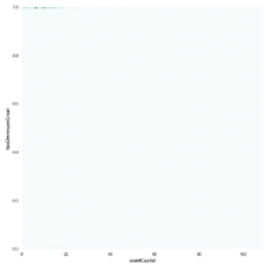
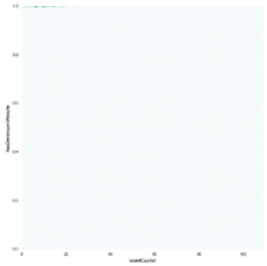
		
`hasDeveloperEmail`	Developer email	
`avgRating`	average app Rating	

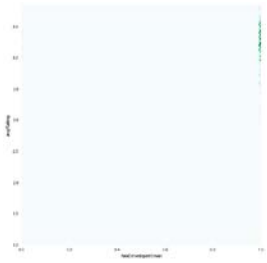
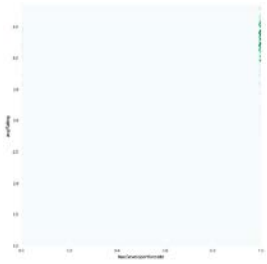
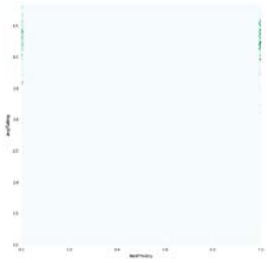
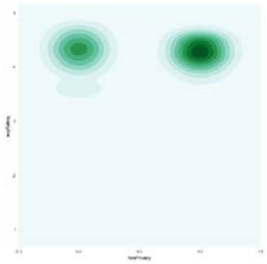
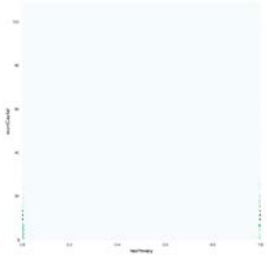
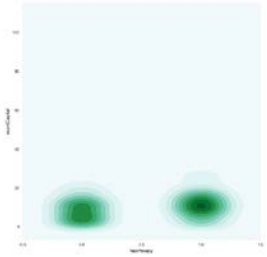
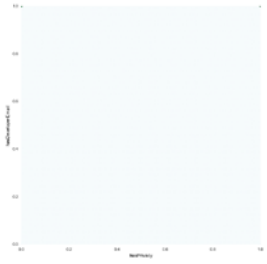
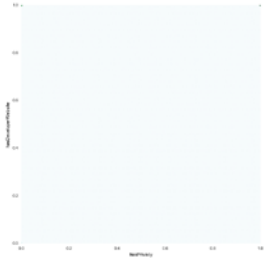
Univariate Conclusion

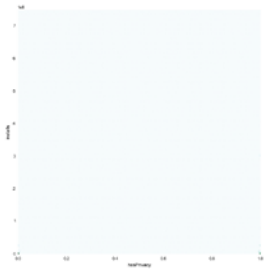
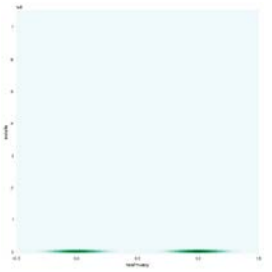
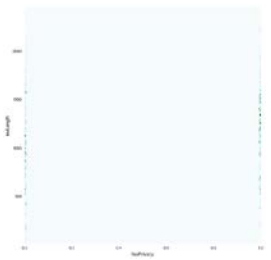
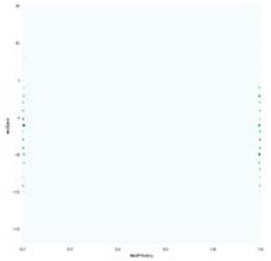
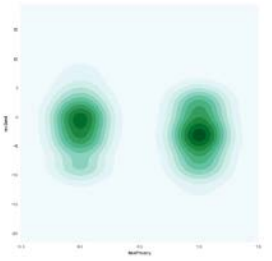
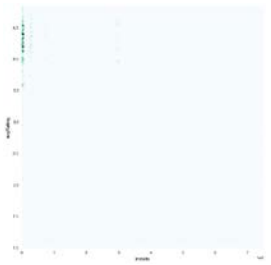
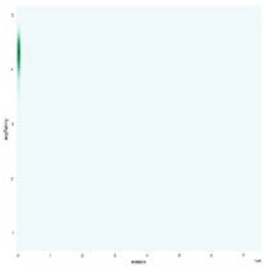
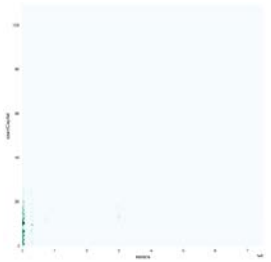
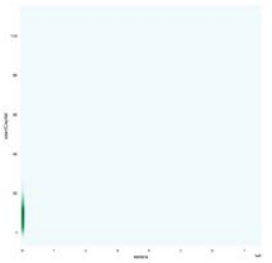
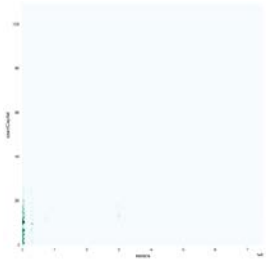
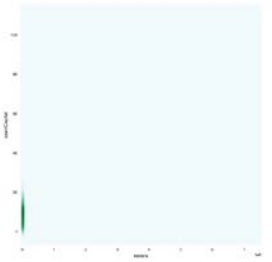
- It is hard to draw an inference about `fair/unfair` but just looking at 1 feature alone.
- `adjectiveCount` tapers in both `fair` and `unfair` for large number of adjectives, so with higher adjectives it might seem it is more likely to be unfair
- `hasPrivacy` is almost evenly distributed for both `fair/unfair`
- `revLength` seems to be uniform in `unfair` apps
- `revSent` for `fair/unfair` distribution is quite similar
- `avgRating` is generally skewed towards higher values for both `fair/unfair`

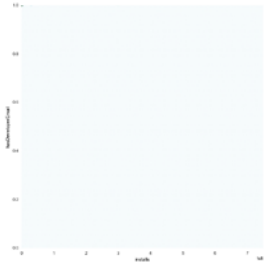
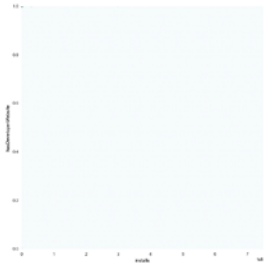
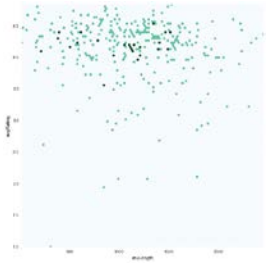
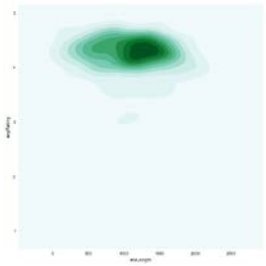
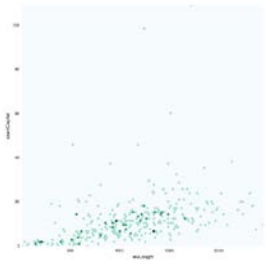
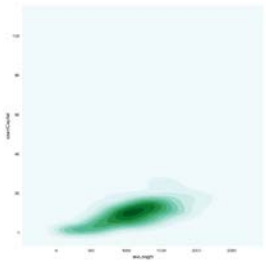
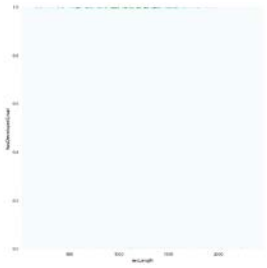
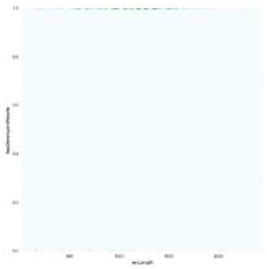
— Bivariate Exploration —

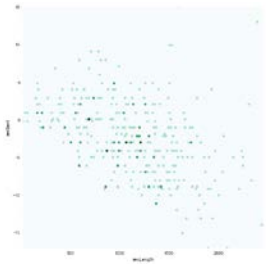
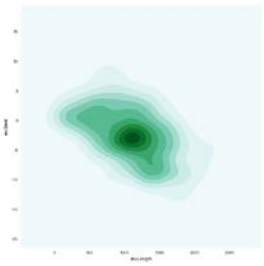
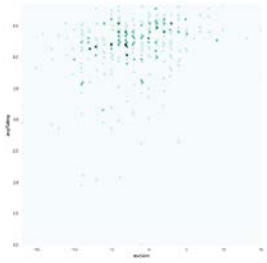
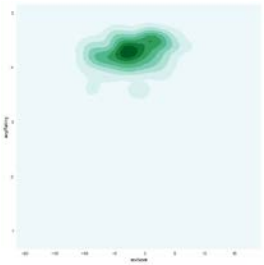
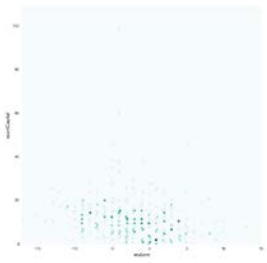
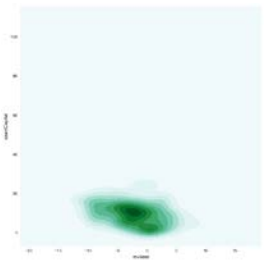
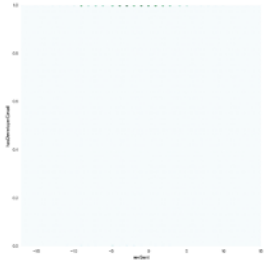
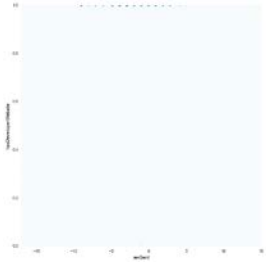
FEATURE1 X FEATURE 2	CORRELATION PLOT (AS HEXAGON BINS)	DENSITY PLOTS (KDE)
`adjectiveCount` x `hasPrivacy`		
`adjectiveCount` x `countCapital`		
`adjectiveCount` x `hasDeveloperEmail`		
`adjectiveCount` x `hasDeveloperWebsite`		
`adjectiveCount` x `hasPrivacy`		
`adjectiveCount` x `installs`		

		
`adjectiveCount` x `revLength`		
`adjectiveCount` x `revSent`		
`countCapital` x `avgRating`		
`countCapital` x `hasDeveloperEmail`		
`countCapital` x `hasDeveloperWebsite`		
`hasDeveloperEmail` x `avgRating`		

		
`hasDeveloperWebsite` x `avgRating`		
`hasDeveloperWebsite` x `hasDeveloperEmail`		
`hasPrivacy` x `avgRating`		
`hasPrivacy` x `countCapital`		
`hasPrivacy` x `hasDeveloperEmail`		
`hasPrivacy` x `hasDeveloperWebsite`		

`hasPrivacy` x `installs`		
`hasPrivacy` x `revLength`		
`hasPrivacy` x `revSent`		
`installs` x `avgRating`		
`installs` x `countCapital`		
`installs` x `countCapital`		
`installs` x `hasDeveloperEmail`		

		
`installs` x `hasDeveloperWebsite`		
`revLength` x `avgRating`		
`revLength` x `countCapital`		
`revLength` x `hasDeveloperEmail`		
`revLength` x `hasDeveloperWebsite`		
`revLength` x `revSent`		

		
`revSent` x `avgRating`		
`revSent` x `countCapital`		
`revSent` x `hasDeveloperEmail`		
`revSent` x `hasDeveloperWebsite`		

Conclusion

- point 1
- point 2

— Classifier Output —

Splitting Dataset into Equal `Fair/Unfair` ratio

Our process:

- split the entire app sample into `fair` apps (**300**) and `unfair` apps (**23**)
- split the fair apps sample into splits of the size of the `unfair` apps.
    - Total **13** splits
- Combined each fair app split with the unfair apps, to make the sample set for classification
- randomly shuffled the classification sample
- trained on `n_sample=36` apps and tested on `total-n_sample = 10` apps for each split
- Calculated classifier reports for each split

SPLIT #	ALGORITHM	AVG PRECISION	AVG ACCURACY (ADJUSTED)	CONFUSION MATRIX ⌈⌈TP FN⌉⌈FP TN⌉⌉
`0th`	kNN `(wt = distance)`	`0.50`	`0.70`	`[[2 3] [2 3]]`
`0th`	kNN `(wt = uniform)`	`0.50`	`0.70`	`[[2 3] [2 3]]`
`0th`	GaussianNB	`0.60`	`0.80`	`[[3 2] [2 3]]`
`0th`	DecisionTreeClassifier	`0.50`	`0.80`	`[[3 2] [3 2]]`
`0th`	RandomForest (`AdaBoostClassifier`)	`0.60`	`0.80`	`[[3 2] [2 3]]`
`0th`	SVM-linear (`SVC`)	`0.62`	`0.70`	`[[2 3] [1 4]]`
`0th`	SVM-Nonlinear (`NuSVC`)	`0.60`	`0.80`	`[[3 2] [2 3]]`

1st	kNN (wt = distance)	0.75	0.80	[[6 2] [1 1]]
1st	kNN (wt = uniform)	0.80	0.90	[[7 1] [1 1]]
1st	GaussianNB	0.64	1.0	[[8 0] [2 0]]
1st	DecisionTreeClassifier	0.53	0.60	[[4 4] [2 0]]
1st	RandomForest (AdaBoostClassifier)	0.75	0.80	[[6 2] [1 1]]
1st	SVM-linear (SVC)	0.04	0.2	[[0 8] [0 2]]
1st	SVM-Nonlinear (NuSVC)	0.80	0.90	[[7 1] [1 1]]
2nd	kNN (wt = distance)	0.71	0.90	[[4 1] [2 3]]
2nd	kNN (wt = uniform)	0.71	0.90	[[4 1] [2 3]]
2nd	GaussianNB	0.38	0.8	[[3 2] [4 1]]
2nd	DecisionTreeClassifier	0.29	0.70	[[2 3] [4 1]]
2nd	RandomForest (AdaBoostClassifier)	0.29	0.70	[[2 3] [4 1]]
2nd	SVM-linear (SVC)	0.50	0.9	[[1 4] [1 4]]
2nd	SVM-Nonlinear (NuSVC)	0.60	0.80	[[3 2] [2 3]]
3rd	kNN (wt = distance)	0.80	0.90	[[4 1]

				[1 4]]
3rd	kNN (wt = uniform)	0.80	0.90	[[4 1] [1 4]]
3rd	GaussianNB	0.81	1.0	[[5 0] [3 2]]
3rd	DecisionTreeClassifier	0.60	0.80	[[3 2] [2 3]]
3rd	RandomForest (AdaBoostClassifier)	0.40	0.70	[[2 3] [3 2]]
3rd	SVM-linear (SVC)	0.81	0.7	[[2 3] [0 5]]
3rd	SVM-Nonlinear (NuSVC)	0.71	0.80	[[3 2] [1 4]]
4th	kNN (wt = distance)	0.68	0.70	[[4 3] [1 2]]
4th	kNN (wt = uniform)	0.68	0.70	[[4 3] [1 2]]
4th	GaussianNB	0.84	1.0	[[7 0] [2 1]]
4th	DecisionTreeClassifier	0.62	0.60	[[3 4] [1 2]]
4th	RandomForest (AdaBoostClassifier)	0.68	0.70	[[4 3] [1 2]]
4th	SVM-linear (SVC)	0.07	0.30	[[0 7] [1 2]]
4th	SVM-Nonlinear (NuSVC)	0.55	0.50	[[2 5] [1 2]]
5th	kNN (wt = distance)	0.62	0.70	[[4 1] [3 2]]

5th	kNN (wt = uniform)	0.62	0.70	[[4 1] [3 2]]
5th	GaussianNB	0.25	1.0	[[5 0] [5 0]]
5th	DecisionTreeClassifier	0.71	0.80	[[3 2] [1 4]]
5th	RandomForest (AdaBoostClassifier)	0.71	0.80	[[3 2] [1 4]]
5th	SVM-linear (SVC)	0.60	0.80	[[3 2] [2 3]]
5th	SVM-Nonlinear (NuSVC)	0.71	0.90	[[4 1] [2 3]]
6th	kNN (wt = distance)	0.80	0.90	[[5 1] [1 3]]
6th	kNN (wt = uniform)	0.80	0.90	[[5 1] [1 3]]
6th	GaussianNB	0.80	1.0	[[6 0] [3 1]]
6th	DecisionTreeClassifier	0.57	0.60	[[2 4] [1 3]]
6th	RandomForest (AdaBoostClassifier)	0.71	0.80	[[3 2] [1 4]]
6th	SVM-linear (SVC)	0.16	0.40	[[0 6] [0 4]]
6th	SVM-Nonlinear (NuSVC)	0.85	1.0	[[6 0] [2 2]]
7th	kNN (wt = distance)	0.80	1.0	[[4 0] [4 2]]
7th	kNN (wt = uniform)	0.83	1.0	[[4 0]

				`[3 3]]`
`7th`	GaussianNB	`0.16`	`1.0`	`[[4 0]` `[6 0]]`
`7th`	DecisionTreeClassifier	`0.52`	`0.80`	`[[2 2]` `[3 3]]`
`7th`	RandomForest (`AdaBoostClassifier`)	`0.45`	`0.90`	`[[3 1]` `[5 1]]`
`7th`	SVM-linear (`SVC`)	`0.16`	`1.0`	`[[4 0]` `[6 0]]`
`7th`	SVM-Nonlinear (`NuSVC`)	`0.45`	`0.90`	`[[3 1]` `[5 1]]`
`8th`	kNN `(wt = distance)`	`0.63`	`0.90`	`[[1 1]` `[5 3]]`
`8th`	kNN `(wt = uniform)`	`0.63`	`0.90`	`[[1 1]` `[5 3]]`
`8th`	GaussianNB	`0.63`	`0.90`	`[[1 1]` `[5 3]]`
`8th`	DecisionTreeClassifier	`0.42`	`0.90`	`[[1 1]` `[7 1]]`
`8th`	RandomForest (`AdaBoostClassifier`)	`0.56`	`0.90`	`[[1 1]` `[6 2]]`
`8th`	SVM-linear (`SVC`)	`0.04`	`1.0`	`[[2 0]` `[8 0]]`
`8th`	SVM-Nonlinear (`NuSVC`)	`0.56`	`0.90`	`[[1 1]` `[6 2]]`
`9th`	kNN `(wt = distance)`	`0.71`	`0.90`	`[[4 1]` `[2 3]]`
`9th`	kNN `(wt = uniform)`	`0.71`	`0.90`	`[[4 1]` `[2 3]]`

`9th`	GaussianNB	`0.78`	`1.0`	`[[5 0] [4 1]]`
`9th`	DecisionTreeClassifier	`0.22`	`0.90`	`[[4 1] [5 0]]`
`9th`	RandomForest (`AdaBoostClassifier`)	`0.25`	`1.0`	`[[5 0] [5 0]]`
`9th`	SVM-linear (`SVC`)	`0.25`	`1.0`	`[[5 0] [5 0]]`
`9th`	SVM-Nonlinear (`NuSVC`)	`0.78`	`1.0`	`[[5 0] [4 1]]`
`10th`	kNN `(wt = distance)`	`0.92`	`0.60`	`[[5 4] [0 1]]`
`10th`	kNN `(wt = uniform)`	`0.93`	`0.7`	`[[6 3] [0 1]]`
`10th`	GaussianNB	`0.93`	`0.80`	`[[7 2] [0 1]]`
`10th`	DecisionTreeClassifier	`0.91`	`0.30`	`[[2 7] [0 1]]`
`10th`	RandomForest (`AdaBoostClassifier`)	`0.91`	`0.2`	`[[1 8] [0 1]]`
`10th`	SVM-linear (`SVC`)	`0.01`	`0.1`	`[[0 9] [0 1]]`
`10th`	SVM-Nonlinear (`NuSVC`)	`0.92`	`0.5`	`[[4 5] [0 1]]`
`11th`	kNN `(wt = distance)`	`0.43`	`0.80`	`[[2 2] [4 2]]`
`11th`	kNN `(wt = uniform)`	`0.78`	`1.0`	`[[4 0] [5 1]]`
`11th`	GaussianNB	`0.45`	`0.90`	`[[3 1]`

				[5 1]]
11th	DecisionTreeClassifier	0.31	0.80	[[2 2] [5 1]]
11th	RandomForest (AdaBoostClassifier)	0.31	0.80	[[2 2] [5 1]]
11th	SVM-linear (SVC)	0.16	1.0	[[4 0] [6 0]]
11th	SVM-Nonlinear (NuSVC)	0.45	0.90	[[3 1] [5 1]]
12th	kNN (wt = distance)	0.60	0.80	[[4 2] [2 2]]
12th	kNN (wt = uniform)	0.60	0.80	[[4 2] [2 2]]
12th	GaussianNB	0.33	0.90	[[5 1] [4 0]]
12th	DecisionTreeClassifier	0.57	0.90	[[5 1] [3 1]]
12th	RandomForest (AdaBoostClassifier)	0.80	1.0	[[6 0] [3 1]]
12th	SVM-linear (SVC)	0.16	0.4	[[0 6] [0 4]]
12th	SVM-Nonlinear (NuSVC)	0.30	0.80	[[4 2] [4 0]]

For Entire Dataset

- scaled the features on MinMaxScaler
- performed k=4 fold Cross-Validation
- Used same classifiers

FOLD #	ALGORITHM	AVG PRECISION	AVG ACCURACY (ADJUSTED)	CONFUSION MATRIX [[TP FN][FP TN]]
1st	kNN (wt = distance)	0.97	0.95	[[74 4][1 1]]
2nd	kNN (wt = distance)	0.93	0.9875	[[76 1][3 0]]
3rd	kNN (wt = distance)	0.90	0.9875	[[75 1][4 0]]
4th	kNN (wt = distance)	0.79	0.9875	[[66 1] [12 1]]
1st	kNN (wt = uniform)	0.97	0.975	[[76 2][1 1]]
2nd	kNN (wt = uniform)	0.93	1.00	[[77 0][3 0]]
3rd	kNN (wt = uniform)	0.90	1.00	[[76 0][4 0]]
4th	kNN (wt = uniform)	0.79	0.9875	[[66 1] [12 1]]
1st	GaussianNB	0.96	0.9125	[[71 7][1 1]]
2nd	GaussianNB	0.96	0.95	[[73 4][1 2]]
3rd	GaussianNB	0.90	1.00	[[76 0][4 0]]
4th	GaussianNB	0.64	0.5875	[[14 53] [5 8]]
1st	DecisionTreeClassifier	0.95	0.975	[[69 10]

1st	DecisionTreeClassifier	0.95	0.875	[[68 10] [2 0]]
2nd	DecisionTreeClassifier	0.92	0.9125	[[70 7][3 0]]
3rd	DecisionTreeClassifier	0.93	0.8625	[[65 11] [2 2]]
4th	DecisionTreeClassifier	0.82	0.975	[[65 2] [10 3]]
1st	RandomForest (AdaBoostClassifier)	0.95	0.9625	[[75 3][2 0]]
2nd	RandomForest (AdaBoostClassifier)	0.93	0.9875	[[76 1][3 0]]
3rd	RandomForest (AdaBoostClassifier)	0.90	1.0	[[76 0][4 0]]
4th	RandomForest (AdaBoostClassifier)	0.82	0.975	[[65 2] [10 3]]
1st	SVM-linear (SVC)	0.95	1.0	[[78 0][2 0]]
2nd	SVM-linear (SVC)	0.93	1.0	[[77 0][3 0]]
3rd	SVM-linear (SVC)	0.90	1.0	[[76 0][4 0]]
4th	SVM-linear (SVC)	0.70	1.0	[[67 0] [13 0]]
1st	SVM-Nonlinear (NuSVC) [1]	-	-	-
2nd	SVM-Nonlinear (NuSVC) [1]	-	-	-
3rd	SVM-Nonlinear	-	-	-

3rd	SVM-Nonlinear (NuSVC) [1]	-	-	-
4th	SVM-Nonlinear (NuSVC) [1]	-	-	-

Classifier Details

CLASSIFIER NAME	CLASSIFIER PARAMETERS
kNN (wt = distance)	(algorithm=auto, leaf_size=30, metric=minkowski, n_neighbors=3, p=2, weights=distance)
kNN (wt = uniform)	(algorithm=auto, leaf_size=30, metric=minkowski, n_neighbors=3, p=2, weights=uniform)
GaussianNB	default
DecisionTreeClassifier	(compute_importances=None, criterion=gini, max_depth=None, max_features=None, min_density=None, min_samples_leaf=1, min_samples_split=2, random_state=None, splitter=best)
RandomForest (AdaBoostClassifier)	AdaBoostClassifier(algorithm=SAMME, base_estimator=DecisionTreeClassifier(compute_importances=None, criterion=gini, max_depth=1, max_features=None, min_density=None, min_samples_leaf=1, min_samples_split=2, random_state=None, splitter=best), base_estimator__compute_importances=None, base_estimator__criterion=gini, base_estimator__max_depth=1, base_estimator__max_features=None, base_estimator__min_density=None, base_estimator__min_samples_leaf=1, base_estimator__min_samples_split=2, base_estimator__random_state=None, base_estimator__splitter=best, learning_rate=1.0, n_estimators=200, random_state=None)
SVM-linear (SVC)	(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0, kernel=rbf, max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)
SVM-Nonlinear (NuSVC)	(cache_size=200, coef0=0.0, degree=3, gamma=0.0, kernel=rbf, max_iter=-1, nu=0.5, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

1. Some bug in the Non-Linear SVM implementation for all apps needs to be resolved. Currently it exits with an error `ValueError: specified nu is infeasible` ↩