Testing ML Classifiers on Different Binary-Labeled Datasets

Michael Mech

UC San Diego

mmech@ucsd.edu

ABSTRACT:

Binary classification problems are the bread and butter tasks of practicing the applications of different machine learning algorithms. Accounting for many different factors, such as dataset size, dimensionality, and representation of data is important when conducting a comparison of different learning algorithms. I present such a comparison using the machine learning algorithms of logistic regression, k-nearest neighbors, and support vector machine, under such conditions of dataset variance by utilizing 3 datasets of different size, one of which, has been transformed into three different representations. In this setting, I examine the effect of the cross-validation method in hypertuning the parameters and improving the performance of the machine learning models, paying special attention to the test,validation, and training iterations of the accuracy scores.

INTRODUCTION:

The pursuit of creating artificial intelligence has led to a diverse domain of knowledge in different fields, synthesizing and sharing their knowledge, leading to the creation of integrated fields. Some fields such as cognitive science rely on the interdisciplinary expertise of other fields, such as inspiration from neuroscience to model for methods of learning. On the other hand, other approaches have developed independent of the image of the human brain, where more hard-science fields like computer science and mathematics have put their efforts into models not limited by the constraints of the human brain.

Being logically-based, such models have their basis on the use of mathematical functions that take in inputs, and churn out outputs in a way which corresponds to whatever designated task at hand. As such, this means that the task must be converted into a representable medium that coincides with mathematics, be it integers, vectors, or matrices. On the topic of such things, the vastly complex, and hence, multidimensional nature of systems in the real world, yield ever-increasing credit to multivariate calculus, and the concept of approximately representing such systems via systems of linear equations makes way for the use of linear algebra. To add to this, the less "linear" systems of the real world may be better represented via the softer concepts of probability and empirical distributions. Hence, mathematics presents the combination of probability, linear algebra, and multivariate calculus to produce input-output functions that can model the world, and all of this can be simulated by the more applied field of computer science. With computer science, these theoretical models can be tested without the intensive use of resources that would otherwise be consumed in real life experiments, with the advantages of being able to store information, tweak it with more flexibility and less precautions, and repeat an otherwise unfeasible amount of iterations of experiments. With all this in mind, the development of such models end up becoming practical tools in the form of algorithms that take in inputs representative of the real world, which brings us full circle to datasets. Datasets, with their variability size, shape, medium of representation, distribution etc., are invaluable in their quality, or susceptibility to being well-trained within algorithms, but the right question is asking which algorithm, based on certain datasets, is the right one to use. For binary classification problems, or tasks where the dataset's target values only compose of two values, algorithms such as support vector machine, random forests, logistic regression, and even k-nearest neighbors are well-suited for training, tuning, and predicting newer iterations of the data.

Methodology:

Datasets

For the first two datasets, which I will call Titanic, and Citrus, I retrieved them off of kaggle. The Titanic dataset originated from a Kaggle competition that gave the information of Titanic passengers such as their age, sex, height, weight etc., while labeling whether or not they survived. The dataset I got was already modified by a kaggle user such that there were extra columns to allow for better one-hot encoding of the available categorical variables, leading to more strings of zeros within the dataset. Even without the extra dimensions from the one-hot encoding, the dataset still had a moderate amount of features to account for. The number of observations in the Titanic dataset was about 1300, making it a dataset on the smaller end of sizes. Lastly, the classes of labels were unbalanced, in that the number of survivors were much less than the number of deceased.

The Citrus dataset is another binary-labeled dataset that I got from Kaggle where the goal was to predict based on numerical measures of diameter, weight, and colors whether or not some fruit is an orange or a grapefruit. There were only 5 features to account for, so this dataset could be considered of lower complexity, but there were 10,000 observations, 5,000 for each class of grapefruit or orange,making this a perfectly balanced dataset. Because I already had plans to use another dataset as the “large” one, I shuffled this dataset, and slimmed it down to 2700 observations, retaining a roughly equal amount of grapefruits and oranges. I consider 2700 observations to be of a moderate size, so this one is my medium-sized dataset.

	diameter	weight	red	green	blue	name
9953	14.34	241.93	156	80	2	1
3850	9.40	166.91	154	73	2	0
4962	11.53	196.54	148	60	2	0
3886	9.43	167.36	149	72	2	0
5437	9.81	171.52	142	53	2	1

Small sample of the citrus dataset.

My last dataset is one that I decided to collect by myself. As it is derived from reddit, I am going to refer to it as the reddit dataset. Essentially, the task I had in mind was to try and be able to tell from Reddit comments, whether or not someone was an introvert or an extrovert. Hence, with the assumption that the posts/comments in a subreddit would exclusively represent the label of the subreddit I am collecting from, I went to the r/extroverts and r/introverts subreddit, and collected about 2800 extrovert comments, and 3700 introvert comments using imported tools from the PRAW module. This would make the total number of observations about 6500, making this my largest dataset. To transform the text of the comments into pluggable data for an algorithm, I utilized sklearn's count-vectorizer to transform them into a word frequency matrix. Since the matrix stored the vocabulary of all unique words within all comments as the features, I capped the dimensions off at 2500, and applied the process of PCA using latent semantic analysis to make the dimensions only 100(# of principal components).

Dataset	*# of Observations*	*# of Dimensions*
Titanic	1307	28
Citrus	2700	6
WFM(Word Frequency Matrices)	6535	2500
WFM(Word Frequency Matrices)	6535	100

Learning Algorithms

From sklearn, I decided to use logistic regression, k-nearest neighbors, and support vector machines as the models of comparison using the data. I wanted the combination of a soft model binary-classifier in the form of logistic regression, a nonparametric model in k-nearest neighbors, and a hard optimization algorithm in support vector machines.

Hypertuning Parameters

```
GrdSrch = GridSearchCV(linsvm, linsvm_grid,cv=5,n_jobs=6)
RndSrch = RandomizedSearchCV(neigh, neigh_grid, n_iter=4, cv=5,n_jobs=6)
```

Whether using GridSearch or RandomizedSearch to cross-validate, the parameters to tune were marginally different across the classifiers. Logistic Regression gave the option of picking l1 or l2 regularization, be it that the dataset has a significant amount of outliers or not, while different values of the regularization constant C were chosen.

```
logreg_grid=dict(C=uniform(loc=0, scale=4),penalty=['l2', 'l1'])
```

For k-nearest neighbors, distance metrics(or how similarity between points are judged) were parameters that could be tuned, and the k number of neighbors to consider when executing the classifier is obviously tunable.

```
neigh_grid = {'n_neighbors': np.arange(1, 8), 'weights': ['uniform','distance']}
```

Lastly, for support vector machines, I played around with different regularization constants, while using squared hinge-loss, and a default of an l2 penalty.

```
linsvm_grid=dict(C=[.01,0.5,1,4,10])
```

Experiment:

Comparing Partitions

The basic process of applying the models of the dataset was to iterate each dataset through three partitions of training data on each of the three models. The point of doing three different partitions of 20/80,50/50,80/20 for training/test is to see the effect of the number of

observations on both training and test errors. Amongst the three models and 3 datasets(plus PCA), here are the averaged out accuracy scores(including validation) labeled by test proportion partition, and type of accuracy score.

	Logistic Regression	K-Nearest Neighbors	Support Vector Machines
Titanic Dataset			
Citrus Dataset			
WFM(Word Frequency Matrices)			
WFM(PCA)			

Titanic Dataset

	Training	Validation	Test
.8	0.708812	0.703556	0.678458
0.5	0.709546	0.697369	0.699286
0.2	0.719936	0.712281	0.703562

	Training	Validation	Test
.8	0.835249	0.740905	0.737094
0.5	0.821848	0.740200	0.755861
0.2	0.893780	0.750558	0.777354

	Training	Validation	Test
.8	0.756066	0.738123	0.746654
0.5	0.670240	0.677037	0.771662
0.2	0.740670	0.746093	0.762087

Citrus Dataset

	Training	Validation	Test
.8	0.935185	0.935185	0.921759
0.5	0.930617	0.930617	0.924198
0.2	0.929630	0.929630	0.927160

	Training	Validation	Test
.8	0.964198	0.964198	0.907716
0.5	0.980988	0.980988	0.909630
0.2	1.000000	1.000000	0.912346

	Training	Validation	Test
.8	0.929012	0.929012	0.918519
0.5	0.826173	0.826173	0.921235
0.2	0.926389	0.926389	0.920988

WFM(Word Frequency Matrices)

	Training	Validation	Test
.8	0.842160	0.700238	0.728374
0.5	0.819463	0.721559	0.709056
0.2	0.809248	0.710591	0.724103

	Training	Validation	Test
.8	0.755626	0.626614	0.629769
0.5	0.847389	0.627549	0.610911
0.2	0.734240	0.625451	0.627639

	Training	Validation	Test
.8	0.852438	0.701911	0.725620
0.5	0.814655	0.722626	0.708457
0.2	0.804810	0.709422	0.720825

WFM(PCA)

	Training	Validation	Test
.8	0.718450	0.718450	0.705498
0.5	0.732363	0.732363	0.696598
0.2	0.725464	0.725464	0.701354

	Training	Validation	Test
.8	0.797792	0.797792	0.642992
0.5	0.724406	0.724406	0.632537
0.2	0.724277	0.724277	0.619288

	Training	Validation	Test
.8	0.715377	0.715377	0.699136
0.5	0.731407	0.731407	0.695451
0.2	0.722455	0.722455	0.700780

Comparing Models

After seeing the effects of different partitions on the different types of accuracy scores in different models and datasets, we can average out the scores for the partitions to easier compare the models. Here is the simplified table of accuracy scores labeled by type of score and type of model:

Kaggle

	Train	Validation	Test
LogReg	0.712765	0.704402	0.693769
KNN	0.850292	0.743888	0.756770
SVM	0.722325	0.720417	0.760134

Reddit

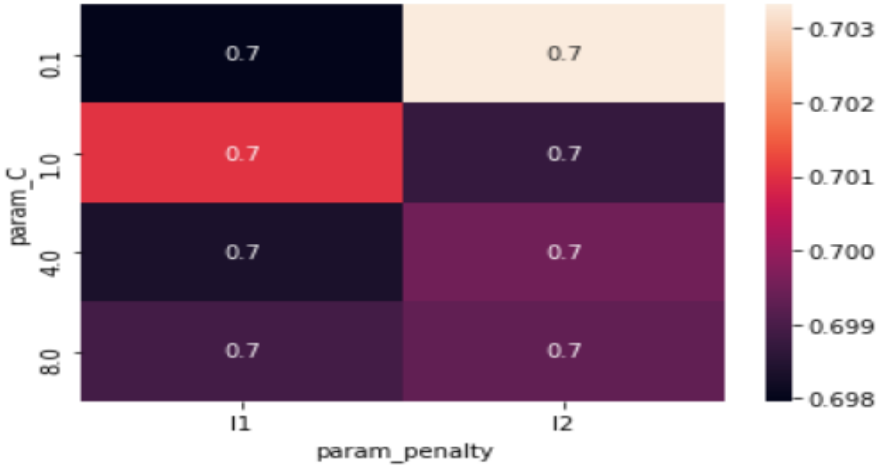
	Train	Validation	Test
LogReg	0.823624	0.710796	0.720511
KNN	0.779085	0.626538	0.622773
SVM	0.823968	0.711320	0.718301

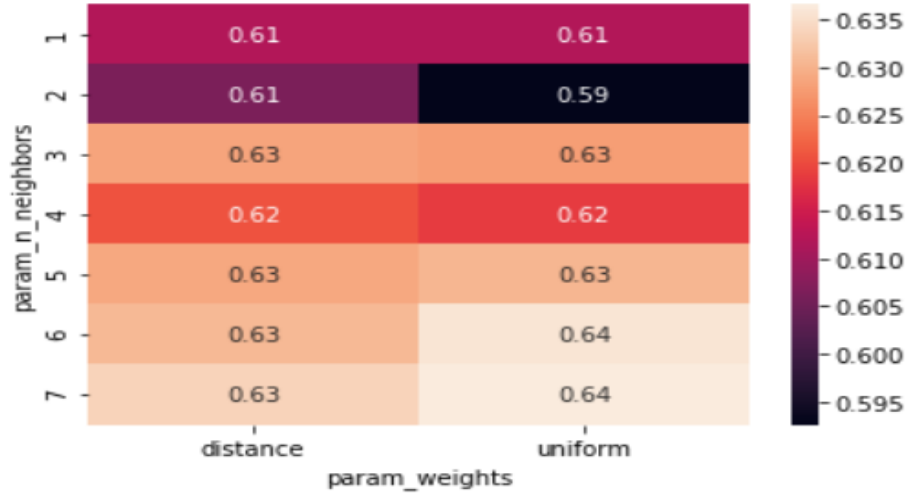
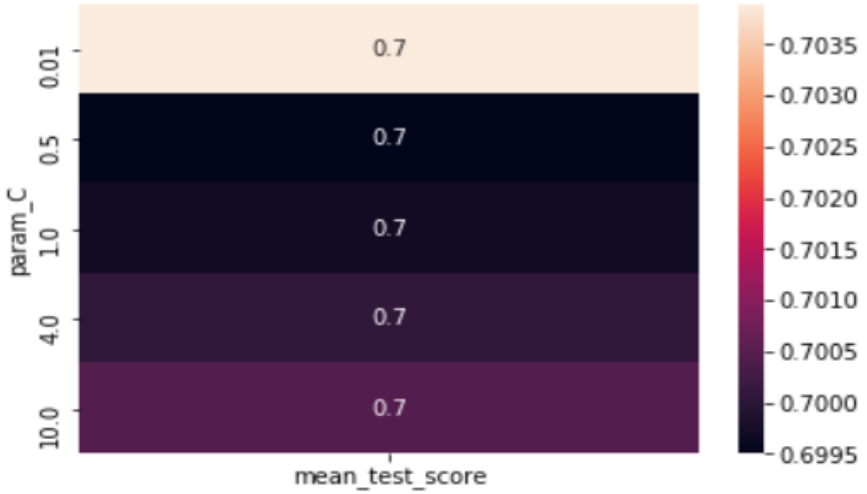
	Train	Validation	Test
LogReg-Citrus	0.930509	0.928900	0.924931
KNN-Citrus	0.986157	0.914317	0.910845
SVM-Citrus	0.887535	0.916331	0.920405

	Train	Validation	Test
LogReg-LSA	0.726395	0.703198	0.702528
KNN-LSA	0.753484	0.638236	0.631893
SVM-LSA	0.724616	0.701783	0.699855

A bit on Variance and Hyperparameters:

Due to the more exhaustive nature of gridsearch over randomizedsearch, the accuracy scores illustrated in the following heatmaps are only shown for the reddit data, where we used gridsearch for cross-validation(for the kaggle datasets we used randomizedsearch). We also report the variance for the reddit data as the average of the standard deviation test scores.

Logistic Regression	Average std_test_scores=0.020717477666727085 

K-Nearest Neighbors	Average std_test_scores=0.011804987945083905 
Support Vector Machines	Average std_test_scores=0.020067956533122364 

Conclusion:

In observing the effects of partitioning data, we would expect that test scores steadily increase when more test data is available. For the titanic data set, this is marginally seen as the test accuracy increases by one percent when the proportion of test data is increased for logistic regression. The effect is drastically seen in k-nearest neighbors as the training accuracy jumps up by six percent when training data jumps from being 20 percent to 80 percent of the overall dataset. The same drastic effect is observed in the same algorithm for the citrus data, where the 80/20 train-test split yielded a 100 percent training score. However, this is where the observed expected pattern ends. For the reddit datasets, the partitions did not do much to improve the training accuracy. Compared to the other datasets ,this would make sense, as the combination of greater complexity and variance of the data would mean that introducing more of the data for the models to train would not necessarily improve them, and possibly even confuse them more. This confusion was only slightly reversed when using the principal components of the reddit data in an effort to use the parts of the data which convey most variance. Interestingly, support vector machines, for all datasets, failed to adhere to the expected trend, with the 50/50 partitions yielding even lower test scores than the 20/80 ones for all three datasets.

The same is expected of test accuracies: with more training data, the test accuracies are expected to increase.This is because adding more data decreases generalization error, decreasing the chances of overfitting, and increasing the test accuracy. For the most part, this has been true, with the test accuracies either increasing in small amounts along with the proportion of training data, or they have remained relatively constant.

```
Best model for Titanic train dataset is ['KNN']
Best model for Titanic validation dataset is ['KNN']
Best model for Titanic test dataset is ['SVM']
Best model for Citrus train dataset is ['KNN-Citrus']
Best model for Citrus validation dataset is ['LogReg-Citrus']
Best model for Citrus test dataset is ['LogReg-Citrus']
```

Purely based off of highest accuracies, for the titanic dataset, K-Nearest Neighbors seems to be the best model, and I hypothesize that the unbalanced nature of the data, along with the unsupervised method of learning from KNN gave it advantages over the other algorithms.

For the citrus dataset, logistic regression did best overall, with SVM trailing with similar scores. Logistic Regression's affinity for the binary classification problem, especially with a super balanced dataset such as the citrus one, shows itself in this instance. Interestingly, the highest training score for the dataset came from KNN, with a whopping 96 percent training accuracy with only 20% training data. KNN seems to do great on such a small partition for all datasets, save for the word frequency model, which negates it with its high variance and complexity.

```
Best model for sparse train dataset is ['SVM']
Best model for sparse validation dataset is ['SVM']
Best model for sparse test dataset is ['LogReg']
Best model for LSA train dataset is ['KNN-LSA']
Best model for LSA validation dataset is ['LogReg-LSA']
Best model for LSA test dataset is ['LogReg-LSA']
```

In the reddit word frequency dataset, it's essentially a tie between SVM and logistic regression for who is the better model, as they have almost practically identical scores for all three accuracy measures. Thus, it does not come to much of a surprise that the same tie is seen when applying PCA to the data. The only comparative increase with the use of PCA comes with KNN, which is the only algorithm that did not see a decrease in accuracy scores once PCA was applied.

All this really shows that machine learning algorithms really depend on the task and data at hand. KNN works best for datasets that do not have much variance, complexity, and are hurt less by imbalance in the number of observations, and the number of observations themselves. Logistic regression does superbly on two-class datasets that are balanced, works well with some variance. SVM seemed to be the most well-rounded, being able to work under all varied conditions of dimensionality, variance, balance and number of observations.

Bonus:

I used a tool called Doc2Vec which transforms a body of text into a representation that expresses relationships between the unique words of the text. Word embeddings are created by trying to figure out relationships between a word and its surrounding words(context), and other way around: context and the word that goes with it. Neural networks are trained to map these relations probabilistically, and the end result are word vectors that can be used inside of a classifier. Stop words and punctuation were removed to reduce noise.

```
           Train  Validation      Test
LogReg  0.833221    0.828691  0.833130
KNN     0.849302    0.758281  0.756420
SVM     0.832547    0.824784  0.830814
Best model for train dataset is ['KNN']
Best model for validation dataset is ['LogReg']
Best model for test dataset is ['LogReg']
```

I trained logistic regression, KNN, and SVM using this newly represented data and am not surprised to see that SVM and logistic regression score similarly once again, as they did for the word frequency and PCA versions of this. Unlike these other versions though, both test and training accuracies were above 80 percent for SVM and logistic regression, meaning that this dataset generalized much better. It would seem that the info conveyed from the relations of text within a document was the crucial factor in attaining an acceptable test score when compared to merely counting the frequency of words. Lastly, I also trained a random forest classifier for fun.

```
Training score for RandFor is: 0.999597747385358
Best Validation score for RandFor is 0.8131536604987932
Tuned test score for RandFor is: 0.8279742765273312
```

References

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd International Conference on Machine Learning - ICML 06*. doi: 10.1145/1143844.1143865