Empirical Study of the Performance of Six Algorithms on Five Binary Classification Problems

Maximilian Siemers siemersm@gmail.com
Department of Cognitive Science
University of California, San Diego
San Diego, CA 92117, USA

Class: COGS 118A: Introduction to Machine Learning I
Instructor: Prof. Zhuowen Tu

Abstract

Using five-fold cross-validation, three different train-test splits of the datasets, and three random dataset shuffles, the algorithms KNN, Logistic Regression, Decision Trees, Bagged Trees, Boosted Trees, and Random Forest were trained on five different data set problems. The influence of several hyperparameters' values on prediction accuracy was visualized and, using the best hyperparameter value combination in each loop, testing accuracy was determined. Averaging over the random shuffles of the data sets, the observed overall ranking order of the classifiers was almost identical to that reported in existing literature.

Word Count: ≈ 2300

1. Introduction

In the realm of machine learning algorithms, there are two kinds of parameters to be distinguished. First, there are the parameters that the algorithm seeks to learn from the data that it is trained on. These parameters are often called weights and determine the algorithm's transformation from the data input space to the prediction outcome space. Then, on a more abstract layer, there are the hyperparameters, which determine *how* the algorithm learns from the data. They often constrain the learning process, assign weight to different loss functions, or influence tweaking strategies such as pruning or regularization.

In machine learning practice (specifically in UCSD COGS 118A homework assignments), hyperparameters are usually either left at their default values or determined by exhaustive grid search. This approach leaves little space for understanding the role of the hyperparameters in influencing the algorithm's fitting quality. Moreover, it might be interesting to investigate whether there exists a systematic relationship between algorithm hyperparameter values, the problem data set used, and the overall prediction accuracy of the algorithm. Getting more insight into this relationship might be helpful to make the search for optimal prediction algorithms both more efficient and automated. For an example approach of automated hyperparameter tuning by applying machine learning to machine learning, see the paper *Learning to learn by gradient descent by gradient descent* from Google DeepMind (Andrychowicz et al., 2016).

Before this background is the motivation of the present study: to visualize and better understand how several binary classifiers' hyperparameters influence training and testing accuracy on different data sets. The goal is to use cross-validation, several different train-test-splits and multiple random samples from the datasets to achieve reliable results. In addition, the average testing accuracies of all classifiers on each of the problem data sets were compared to find out whether they could replicate the results from the comprehensive empirical classification performance study by Caruana and Niculescu-Mizil (2006).

2. Method

2.1 Problems

Five supervised binary classification problem data sets that are available at the UCI Machine Learning database (Dheeru and Karra Taniskidou, 2017) were used for this analysis: The Covertype Data Set (*COVTYPE*), the Census Income Data Set (*INCOME*), the Letter Recognition Data Set (*LETTER*), the Iris Data Set (*IRIS*), and the Breast Cancer Wisconsin (Original) Data Set (*WDBC* Mangasarian and Wolberg, 1990). All of these five data sets consist of rows with sample observations and columns with features. The binary classification target was either included in the data set download (INCOME, IRIS, WDBC) or created from the existing multi-class target (COVTYPE, LETTER).

The following paragraphs describe the purposes of each of the five data sets and gives examples of their features.

COVTYPE Predictors are quantitative and qualitative measurements of cartographic variables for 30x30 meter cells from the US Geological Survey (USGS) and US Forest Service (USFS) data. The outcome is a class of seven forest cover types as determined by the USFS Region 2 Resource Information System (RIS) data. The largest outcome class is interpreted as a positive outcome. Predictor examples: *Elevation* measured in meters, *Soil Type* as indicated by one of 40 classes.

INCOME Predictors are quantitative and qualitative US Census data. The outcome is the yearly income of a given subject, indicated by either >$50k/y or <$50k/y. The first is interpreted as a positive outcome, the second as a negative. Predictor examples: *Age* in years, *Relationship* as one of the class Wife, Own-Child, Husband, Not-in-family, Other-relative, Unmarried.

LETTER Predictors are quantitative numerical attributes (statistical moments and edge counts) of a large number of black-and-white rectangular pixel displays of the 26 capital letters in the English alphabet. The outcome is the English letter, where letters A-L were used as a positive and letters M-Z were used as a negative outcome. Predictor examples: mean x y correlation, correlation of y-edges with x.

IRIS Predictors are sepal and petal width and length of iris plants. The outcome is the class of three types of iris plants. The first class (Iris Setosa) was used as a negative outcome, the second (Iris Versicolor) as a positive outcome.

WDBC Predictors are quantitative measurements of attributes of breast cancer samples. The outcome is the class of benign and malignant diagnosis results, where benign was interpreted as a negative and malignant as a positive outcomes. Predictor examples: Clump Thickness (1-10), Uniformity of Cell Shape (1-10). The data stem from the University of Wisconsin Hospitals.

In order for the analysis based on heavily nested loops to run in sensible time, the number of observations in all datasets with $|DATASET| > 2000$ was limited to 2000 randomly drawn samples. Table 1 lists the resulting and original data set sizes.

2.2 Classifier Algorithms

Six classification algorithms were trained on the data sets described above: K Nearest Neighbors (*knn*), Logistic Regression (*logreg*), Decision Trees (*dt*), Bagged Decision Trees (*bagdt*), Boosted Decision Trees (*bstdt*), and Random Forest (*rf*). All were implemented in Python using the corresponding sub-package from the *scikit-learn* package. The following paragraphs describe the hyperparameters that were varied in grid search and in analyzing their influence on classification performance.

Table 1: Original and limited sizes of data sets

Data set	**Orig. Size**	**Lim. Size**
COVTYPE	(581011×55)	(2000×55)
INCOME	(32561×109)	(2000×109)
LETTER	(20000×16)	(2000×16)
IRIS	— (150×5) —	
WDBC	— (569×31) —	

K Nearest Neighbors (knn) K was varied to assume all values between $K = 1$ and $K = 30$, and, if $|DATASET| > 30$, 10 additional values that were evenly spread between 30 and $|DATASET|$. If $\frac{|DATASET|}{cv\_fold} < 30$, the values of K assumed all values up to that maximal value.

Logistic Regressoin (logreg) The inverse regularization parameter C was varied between 10^{-8} and 10^{4} in factors of 10. Larger values of C allow the parameters of the model to vary more freely.

Decision Tree (dt) Two Decision Tree hyperparameters were varied: *max_depth* determines the maximum depth that a tree is allowed to grow and was varied to assume values between 1 and 29. *max_features* determines how many features are considered when finding the best split and was also varied between 1 and 29 (or $n_{features}$ in the data set if $n_{features} < 29$) in steps of 1.

Bagged Decision Trees (bagdt) The number of Decision Tree estimators in the ensemble *n_estimators* was varied to assume values between 1 and 29 in steps of 1.

Boosted Decision Trees (bstdt) Same manipulations as in Bagged Decision Trees.

Random Forest (rf) *max_features* determines how many features are considered when finding the best split at each tree in the forest and was varied to assume all values in $\{1, 2, 4, 6, 8, 12, 16, 20\}$ that were smaller or equal to $n_{features}$. The number of trees in the forest *n_estimators* was set to 1024.

3. Experiments

Each of the five data sets (see Table 1) was divided into training and testing sets for each of the relative training sizes .5, .2, and .8 (*train splits*). In order to exclude the random sampling procedure as a potential confounding cause of results, the splits into training and testing sets were repeated three times each (*shuffles*) with varying random seeds, thus resulting in nine analysis loops.

Within each loop, an exhaustive grid search for all combinations of hyperparameters (see Section 2.2) of all classifiers was run on each problem data set. The parameter search was conducted using five-fold cross-validation. The averaged cross-validated prediction accuracies for each algorithm, data set, train split run, and random shuffling run were saved for later analysis.

3.1 Prediction Accuracy by Hyperparameter

Figures 1 and 2 depict the relationships between hyperparameter values and cross-validated prediction accuracy score. The performance metrics are averaged across problems, shuffles, and train splits. For Decision Trees, where two different hyperparameters were varied, each of their corresponding plots shows performance averaged over the varying values of the other hyperparameter.

3.2 Train/Test split and Performance

The influence of the train split and the resulting relative sizes of training and testing sets on prediction accuracy is shown separately for training and testing accuracies in Figure 3.

3.3 Overall Classifier Performance

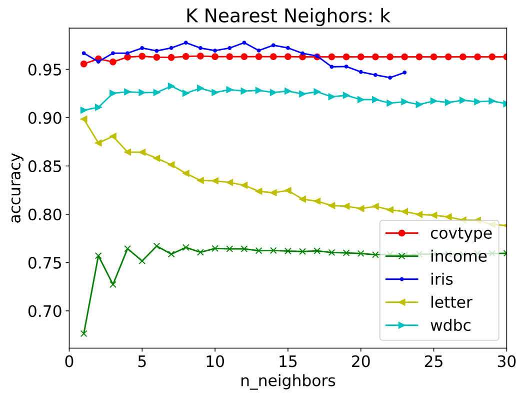
The estimates of overall classifier performance are based on the best hyperparameter combinations that were yielded by exhaustive grid search for each shuffle for *train split* = .8. Table 2 lists these optimal hyperparameter values for each classifier, problem data set, and random shuffle. Based on models with these values, the prediction accuracy on the testing set was determined and averaged over the three shuffles. Table 3 indicates the thus obtained mean testing accuracy for each classifier by problem data set.

Finally, testing accuracies were averaged over problem data set. The resulting overall mean testing performances and rank order is given by Table 4 .

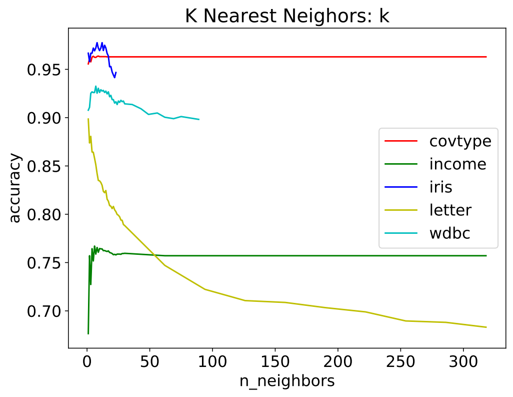
4. Conclusion

The present analysis managed to replicate a subset of the empirical results from Caruana and Niculescu-Mizil (2006). The higher testing accuracies in Table 4 in this study indicate that the problems were easier, but the ranking order is identical except for third and fourth place classifiers KNN and Boosted Trees - however, their results were extremely close to each other.

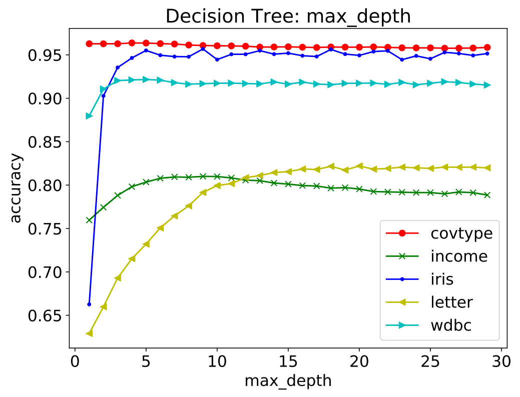
Considering the influence of hyperparameters on prediction accuracy, there seem to be three groups among the classifiers studied here:



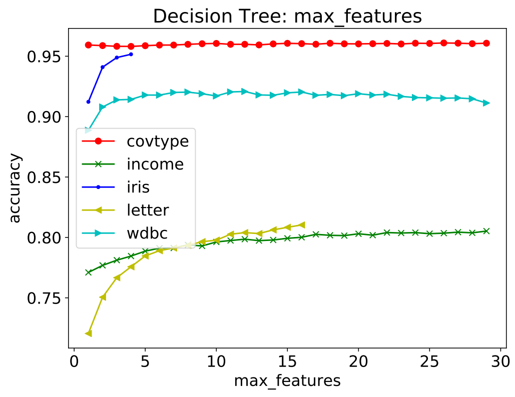
(a) KNN: k



(b) KNN: k



(c) Decision Tree: *max_depth*

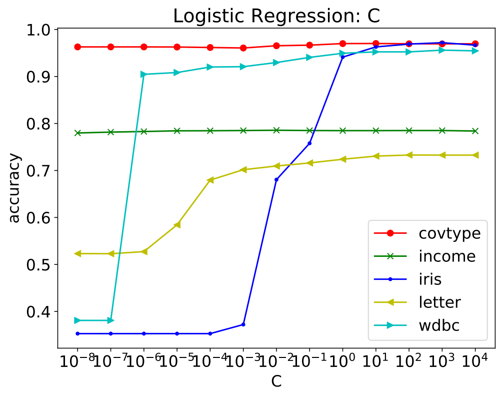


(d) Decision Tree: *max_features*

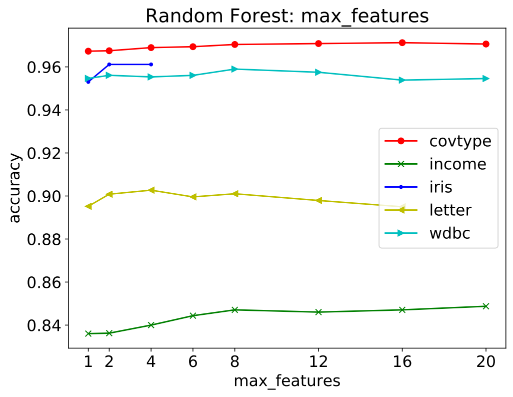
Figure 1: Average cross-validated prediction accuracy by hyperparameter value for KNN (1a and 1b) and Decision Tree (1c and 1d), averaged over problems, shuffles, and train split. (1a) zooms in on a subset of the results shown in (1c) for greater detail. (1c) depicts results for varying levels of *max_depth*, averaged over *max_features*, and vice versa in (1c).

Type 1: Monotonic relationship between hyperparameter and performance
For *max_features* in Decision Trees, testing performance stagnates for values higher than approximately 8 for all problems other than INCOME and LETTER, where testing accuracy kept slowly increasing after that for problems INCOME and LETTER. The regularization parameter C in Logistic Regression showed an interesting influence on testing accuracy, which forms the shape of a sigmoid on the IRIS, LETTER, and WDBC problems. In both Bagged and Boosted Decision Trees, higher values of *n_estimators* correspond with higher testing accuracies.

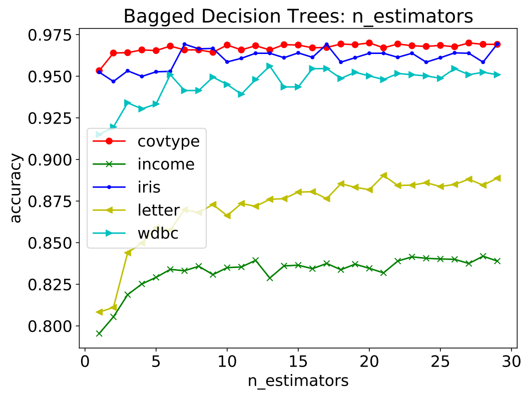
Type 2: Local optimum or quick convergence In Decision Trees, testing accuracy converged quickly towards an upper limit with increasing values of *max_depth*, except for the problem INCOME, where testing accuracy started to slowly decrease after a hyperparameter value of approximately 10. The number K of nearest neighbors considered in KNN



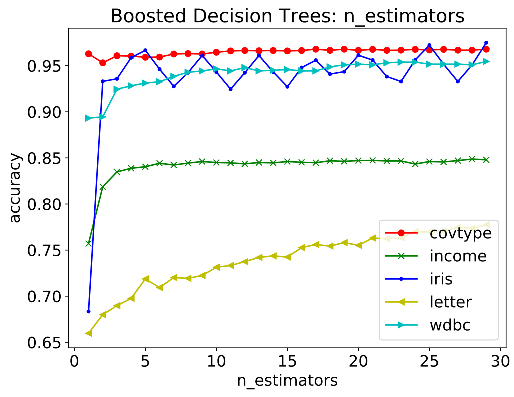
(a) Logistic Regression: C



(b) Random Forest: $n\_estimators$



(c) Bagged Decision Trees: $n\_estimators$



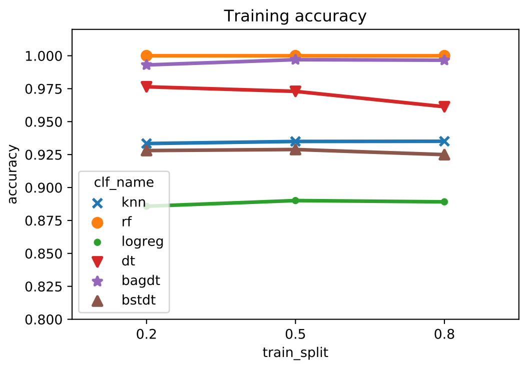
(d) Boosted Decision Trees: $n\_estimators$

Figure 2: Average cross-validated prediction accuracy by hyperparameter value for Logistic Regression (2a), Random Forest (2b), Bagged Decision Trees (2c), and Boosted Decision Trees (2d).

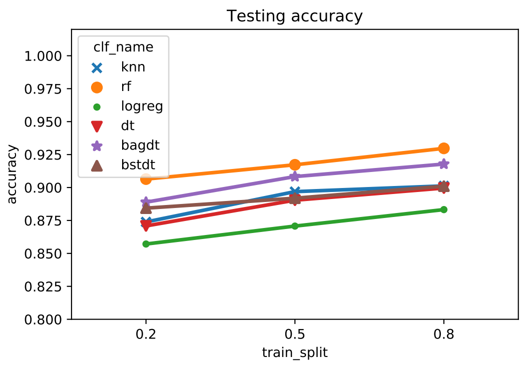
seemed to have a "sweet spot" in form of a local maximum in testing accuracy in the lower range of values, but after an initial steep increase for most problems. A notable exception is the LETTER data set, where lower values of K are better (see also 2).

Type 3: Unclear The relationship between $n\_estimators$ and testing accuracy in Random Forest is less clear. For WDBC, there is a slight positive correlation, whereas for LETTER, it is slightly negative. These results might be due to a ceiling effect, since Random Forest achieved very high overall.

These results seem to imply systematic relationships between hyperparameters and testing accuracy. This makes the further development of Machine Learning algorithms to automatically tune model hyperparameters even more promising. The present study also visualized how much these relationships differ between different data sets, which shows that optimal hyperparameters indeed seem to be a problem that needs to be learned from data.



(a) Training accuracy



(b) Testing accuracy

Figure 3: Training (3a) and testing (3b) accuracy by train split and classifier, averaged over problems and shuffles (rf: Random Forest, logreg: Logistic Regression, dt: Decision Tree, bagdt: Bagged Trees, bstdt: Boosted Trees)

Table 2: Best hyperparameter values after grid search by classifier and problem in each of three random shuffles (0.8 train split):

		Problem				
Classifier	**Hyperparam.**	**COVTYPE**	**INCOME**	**IRIS**	**LETTER**	**WDBC**
bagdt	*n_estimators*	3,27,14	26,25,28	9,29,7	21,14,18	13,9,16
bstdt	*n_estimators*	24,19,12	28,18,18	5,29,24	29,28,24	21,29,10
dt	*max_depth*	3,5,4	8,4,6	9,9,3	18,19,18	6,5,6
	max_features	18,24,19	18,24,26	3,4,4	10,15,6	7,11,15
knn	*n_neighbors*	4,5,9	8,6,17	12,9,1	1,1,1	9,7,7
logreg	*C*	10^2,1,10	10^{-5},01,01	10,10^3,10^2	10^2,10^3,10^2	$10^3$0,10,10^3
rf	*max_features*	20,16,16	20,16,8	1,2,2	4,2,8	12,8,8

5. Bonus Points

The code that this analysis is based on is highly generalized and abstracts over algorithms, data, train splits, and random shuffles. Each of these can conveniently be adjusted in the configuration part of the script, and calling one function is then enough to run all loops required for the present study (siehe code below). New classifiers can be implemented by adding a link to a scikit-learn estimator interface compatible class in *CLF_DICT*. The grid of hyperparameters to be varied is set by entries in *CLF_PARAM_DICT*, using their scikit-learn estimator interface argument names. Similarly, additional data sets can be added by new named (X, y) tuples in DATA_DICT. Any number of train split sizes can be indicated in TRAIN_SPLITS, the maximal data size to be computed is determined by MAX_DATA_SIZE, and the number of randomly drawn sub-samples from the data set is set by N_SHUFFLES. CV is the number of cross-validation folds, and knn_fillup is the number of evenly spaced values of K for KNN that are added to the parameter grid between the highest number indicated in CLF_PARAM_DICT and the maximum valid number determined by $\frac{|DATASET|}{CV}$.

Table 3: Classifier testing accuracy by problem, averaged over shuffles (0.8 train split):

Classifier	**WDBC**	**INCOME**	**IRIS**	**COVTYPE**	**LETTER**
Bagged Trees	.968	.833	.944	.971	.873
Boosted Trees	.968	.840	.956	.971	.771
Decision Tree	.953	.799	.956	.970	.820
KNN	.950	.757	.933	.967	**.899**
Logistic Regression	.968	.792	**.967**	.968	.721
Random Forest	**.982**	**.848**	.944	**.976**	.897

Table 4: Overall average testing accuracy by classifier and comparison with results from Caruana and Niculescu-Mizil (2006)

(a) Ranked classifiers by testing accuracy, averaged over problems and shuffles (0.8 train split):

Rank	**Classifier**	**Accuracy**
1	Random Forest	.930
2	Bagged Trees	.918
3	KNN	.901
4	Boosted Trees	.901
5	Decision Tree	.900
6	Logistic Regression	.883

(b) Results from Caruana and Niculescu-Mizil (2006)

Rank	**Classifier**	**Accuracy**
1	Random Forest	.872
2	Bagged Trees	.846
3	Boosted Trees	.834
3	KNN	.756
5	Decision Tree	.647
6	Logistic Regression	.636

```
CLF_DICT        = {'logreg': linear_model.LogisticRegression(),
                   'knn':    neighbors.KNeighborsClassifier(),
                   'rf':     ensemble.RandomForestClassifier(),
                   'svm':    svm.SVC(),
                   'dt':     tree.DecisionTreeClassifier(),
                   'bagdt':  ensemble.BaggingClassifier(),
                   'bstdt':  ensemble.AdaBoostClassifier()}

CLF_PARAM_DICT = {'knn':    {'n_neighbors':  np.arange(1, 31)},
                  'rf':     {'n_estimators': np.array([1024]),
                  'max_features': np.array([1, 2, 4, 6, 8, 12, 16, 20])},
                  'logreg': {'C': np.array([1e-8, 1e-7, 1e-6, 1e-5, 1e-4,
                  1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4])},
                  'dt':     {'max_depth':    np.arange(1, 30),
                  'max_features': np.arange(1, 30)},
                  'bagdt':  {'n_estimators': np.arange(1, 30)},
                  'bstdt':  {'n_estimators': np.arange(1, 30)}}

DATA_DICT        = {'wdbc':      (wdbc_X, wdbc_y),
                  'income':    (income_X, income_y),
                  'iris':      (iris_X, iris_y),
                  'covtype':   (covtype_X, covtype_y),
                  'letter':    (letter_X, letter_y)}

MAX_DATA_SIZE  = 2000
TRAIN_SPLITS   = [0.2, 0.5, 0.8]
N_SHUFFLES     = 3
CV             = 5
KNN_FILLUP     = 10

loop = BigLoop(DATA_DICT, CLF_PARAM_DICT, TRAIN_SPLITS, N_SHUFFLES, CV,
knn_fillup=KNN_FILLUP, method='grid_search', n_jobs=4)
loop.run()
```

The BigLoop class already has a method included that allows to plot testing accuracy by hyperparameter for a given classifier and hyperparameter, averaged over shuffles and train splits. Another big advantage of this implementation is that the complete analysis, including the data, parameter combinations, results, and involved sklearn objects can be saved to one Pickle file, which can then quickly be loaded for further analysis without having to re-run the whole analysis.

The code that this analysis is based on is completely available on GitHub (`https://github.com/phi1eas/118a_project`), including the scripts that generated all plots and the LaTeX code underlying the tables in this report from experiment results saved in a bigloop.pkl file.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv:1606.04474 [cs]*, June 2016. URL `http://arxiv.org/abs/1606.04474`. arXiv: 1606.04474.

Rich Caruana and Alexandru Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM. ISBN 978-1-59593-383-6. doi: 10.1145/1143844.1143865. URL `http://doi.acm.org/10.1145/1143844.1143865`.

Dua Dheeru and Efi Karra Taniskidou. UCI Machine Learning Repository. Technical report, University of California, Irvine, School of Information and Computer Sciences, 2017. URL `http://archive.ics.uci.edu/ml`.

O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1&18, September 1990.