Supplementary Information (SI) for Digital Discovery.

SUPPORTING INFORMATION FOR DATA EFFICIENCY OF CLASSIFICATION STRATEGIES FOR CHEMICAL AND MATERIALS DESIGN

Quinn M. Gallagher
Chemical and Biological Engineering
Princeton University
Princeton, NJ 08544
qg1361@princeton.edu

Michael A. Webb
Chemical and Biological Engineering
Princeton University
Princeton, NJ 08544
mawebb@princeton.edu

Contents

S1 Model Implementation and Hyperparameter Tuning

Random forests were implemented using the `RandomForestClassifier` class from the `scikit-learn` Python package. Features were kept unscaled when training. Uncertainties were calculating by taking the entropy (*i.e.*, $-\sum_i p_i \ln p_i$) of the class distribution produced by the `predict_proba` method. Extreme gradient-boosted decision trees (XGBs) were implemented using the `XGBClassifier` method from the `xgboost` Python package. Features were kept unscaled when training. Uncertainties were calculating by computing the standard deviation in predictions produced by training 10 models on 70% subsamples of the training set. Neural networks were implemented using the `MLPClassifier` class from the `scikit-learn` Python package. Features were scaled using a `MinMaxScaler` before training. Uncertainties were calculated by taking the standard deviation in predictions produced by training 10 models on 70% subsamples of the training set. Support vector classifiers were implemented using the `SVC` class from `scikit-learn`. Features were scaled using a `MinMaxScaler` before training. Uncertainties were calculated using the entropy of the probability distribution computed by the `predict_proba` method. LP models were implemented using the `LabelPropagation` class from `scikit-learn`, features were scaled using a `MinMaxScaler` before training, and uncertainties were calculated using $1 - p_{\text{max}}$, where p_{max} is the probability of the most likely class predicted by the `predict_proba` method.

For all models implemented using `scikit-learn` or `xgboost`, hyperparameters were tuned using the `BayesSearchCV` method from the `scikit-optimizer` Python package, which uses a sequential, GP-based Bayesian optimization protocol for hyperparameter selection. For a set of hyperparameters, performance was gauged using the Macro F_1 score computed by 5-fold cross-validation. A maximimum of 100 iterations was conducted, but if the Macro F_1 score did not improve by 0.03 in 10 iterations, hyperparameter tuning was terminated. For each model, the hyperparameters chosen for tuning and their available ranges are available for inspection in the GitHub available at `https://github.com/webbtheosim/classification-suite.git`.

All Gaussian processes (GPs) were implemented in the `gpytorch` Python package. Gaussian process regressors (GPRs) were implemented using `GaussianLikelihoods` with a `MarginalLogLikelihood` loss function, and Gaussian process classifiers (GPCs) were implemented using `BernoulliLikelihoods` with a `VariationalELBO` loss function. Isotropic GPs used radial basis function kernels with a single lengthscale, while anisotropic GPs used radial basis functions with d lengthscales, where d is the dimensionality of the task. Features were scaled using a `MinMaxScaler` prior to training. Hyperparameters, including lengthscales, were fit using the `Adam` optimizer until loss functions converged within 1e-6.

The Bayesian kernel density estimation (BKDE) model was adapted from the Gryffin GitHub available at `https://github.com/aspuru-guzik-group/gryffin.git`. Kernel density estimation remained unchanged from the official implementation. The Bayesian autoencoder used for the BKDE model was kept at a fixed architecture of three hidden layers with 24 hidden dimensions, consistent with its use in Ref. [1]. Features are scaled internally by BKDE prior to the training of the Bayesian autoencoder. Predictions are made on a point $\mathbf{x}$ by finding the sum of kernel densities at $\mathbf{x}$ for each label. That is, if $\{\mathbf{x}^0\}$ denotes the set of points labeled 0, $\{\mathbf{x}^1\}$ denotes the set of points labeled 1, $\phi(\mathbf{x}_i)$ denotes BKDE's kernel density estimate of point i, and $\mathbf{X}$ denotes the entire domain, the probability of a new point $\mathbf{x}^*$ being labeled l is $p_l(\mathbf{x}^*) = \sum_{\mathbf{x_i} \in \{\mathbf{x}^l\}} \phi_i(\mathbf{x}^*) / \sum_{\mathbf{x_i} \in \mathbf{X}} \phi_i(\mathbf{x}^*)$. The class with the highest probability is chosen as the predicted label. Uncertainties are computed using the entropy of the probability distribution across all labels.

S2 Sensitivity of Top Algorithms to Chosen Tasks and Number of Points

Figure 3 shows the performances of top-performing algorithms for the maximum number of points measured on every task. In this section, we consider how the top-performing algorithms change when different tasks or number of points are considered. Below, we display the performances of top-performing algorithms for different subsets of tasks based on their dimensionality, d.

Figure S1 shows the performances of top algorithms for tasks where $d \leq 8$. The results in Figure S1 differ from those in Figure 3. SV-based active learning algorithms are more represented among the top-performers in Figure S1 than in Figure 3A, while XGB-based active learning algorithms are no longer top performers for this set of tasks. NN-based algorithms continue to perform well; even NN-based space-filling algorithms are represented among the top performers. Figure S1B shows that NN-based active learning algorithms are still top performers when measuring performance by ξ, while RF-based active learning algorithms perform less well relative to GP-based active learning algorithms.

Figure S2 shows the performances of top algorithms for tasks where $d > 8$. Here, RF- and XGB-based active learning algorithms show increased performance relative to other strategies. When considering the results of both S1 and S2, it seems that the relative performance of tree-based algorithms is highly dependent on the dimensionality of the task. While NN-based algorithms perform well across all dimensions studied here, tree-based methods (*i.e.*, RFs and XGBs) seem to improve with more dimensions, while kernel-based methods (*i.e.*, GPs and SVs) seem to improve with lower dimensions.

We also consider how top-performing algorithms differ when using three, five, and seven rounds of active learning. Here, we consider all tasks. Figure S3 shows that, for few rounds of active learning, there are space-filling algorithms that perform just as well as active learning algorithms. It also seems that RF-based algorithms are less clearly the second best option compared to GPs or SVs. For the results of five and seven rounds of active learning (shown in Figures S4 and S5, respectively), space-filling algorithms decline in their relative performance and the results converge to those shown in Figure 3. These results indicate that NN-based active learning algorithms are still the top performers even when data is scarce.

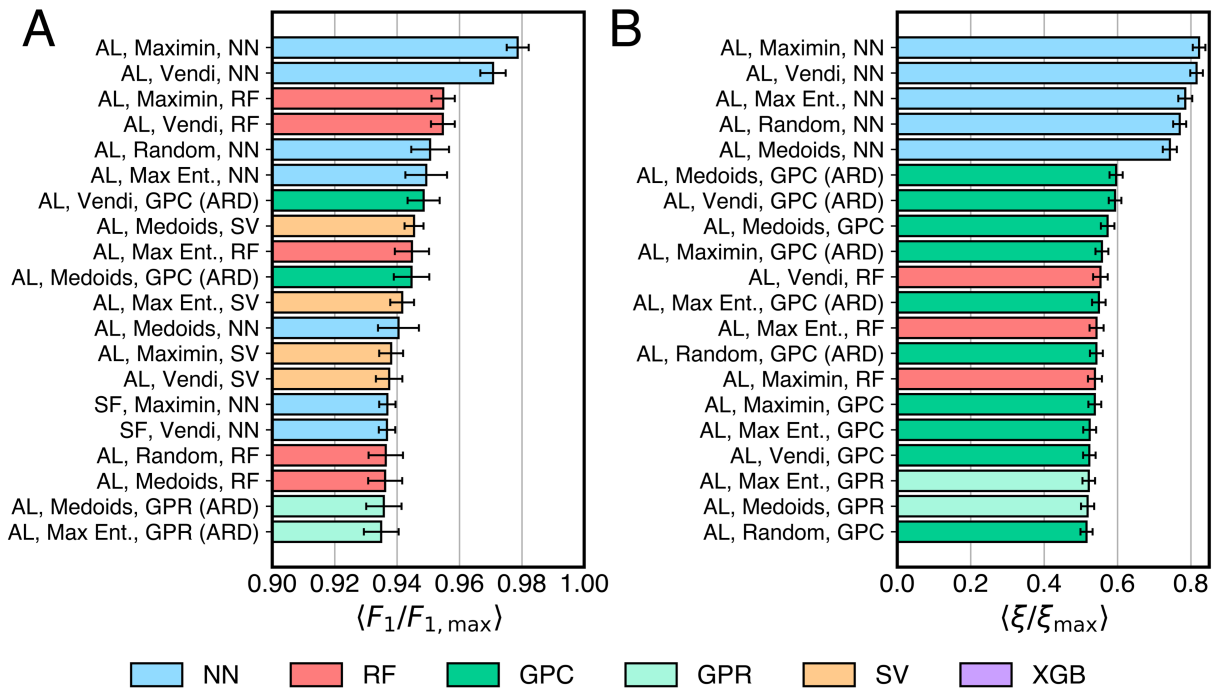

Figure S1: Summary of performances of the top-20 algorithms on tasks with dimensionality d where $d \leq 8$. Algorithm performance is measured by averaging the (A) relative Macro F_1 score and (B) relative ξ of each algorithm across all tasks, where "relative" denotes normalizing the metric by the performance of the top-performing algorithm on that task. Results are colored according to the model used by the specified algorithm. Error bars show the standard error.

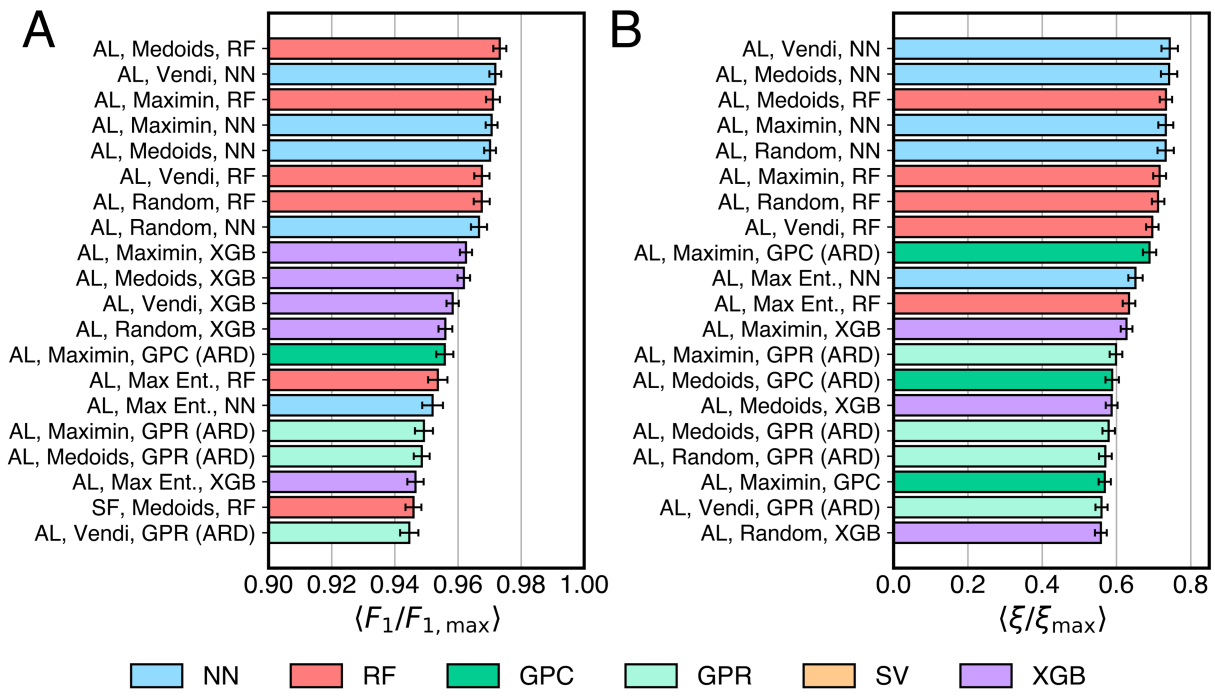

Figure S2: Summary of performances of the top-20 algorithms on tasks with dimensionality d where $d > 8$. Algorithm performance is measured by averaging the (A) relative Macro F_1 score and (B) relative ξ of each algorithm across all tasks, where "relative" denotes normalizing the metric by the performance of the top-performing algorithm on that task. Results are colored according to the model used by the specified algorithm. Error bars show the standard error.

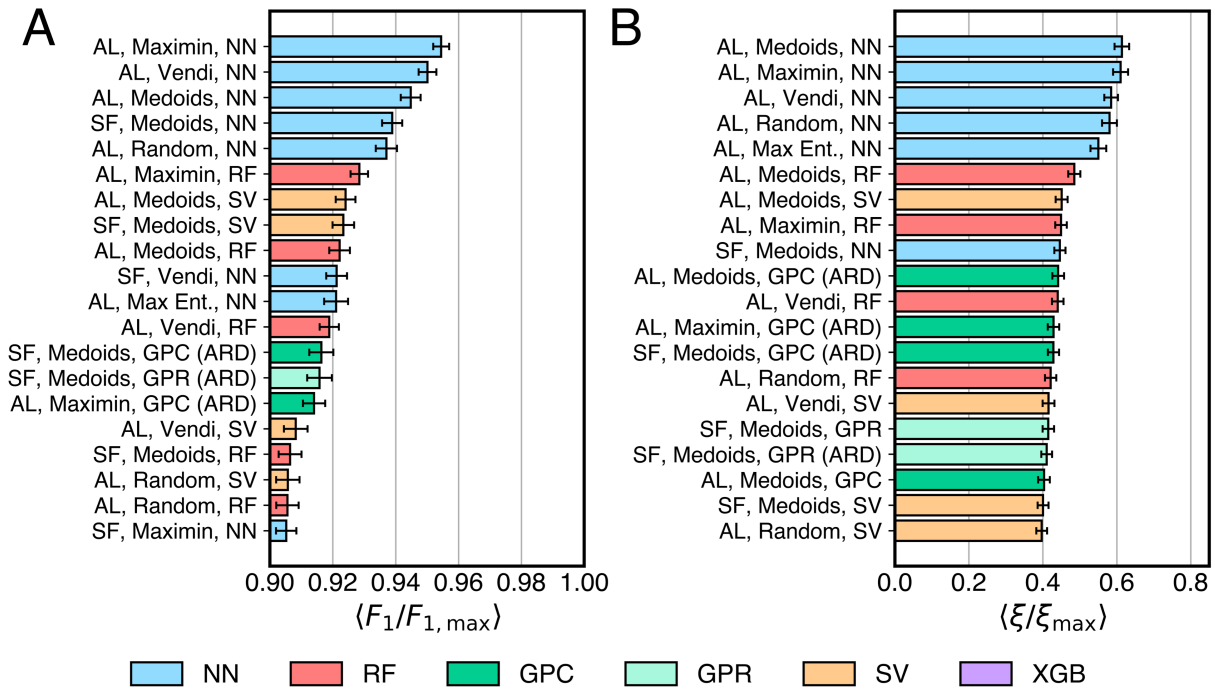

Figure S3: Summary of performances of the top-20 algorithms after three rounds of active learning on all tasks. Algorithm performance is measured by averaging the (A) relative Macro F_1 score and (B) relative ξ of each algorithm across all tasks, where "relative" denotes normalizing the metric by the performance of the top-performing algorithm on that task. Results are colored according to the model used by the specified algorithm. Error bars show the standard error.

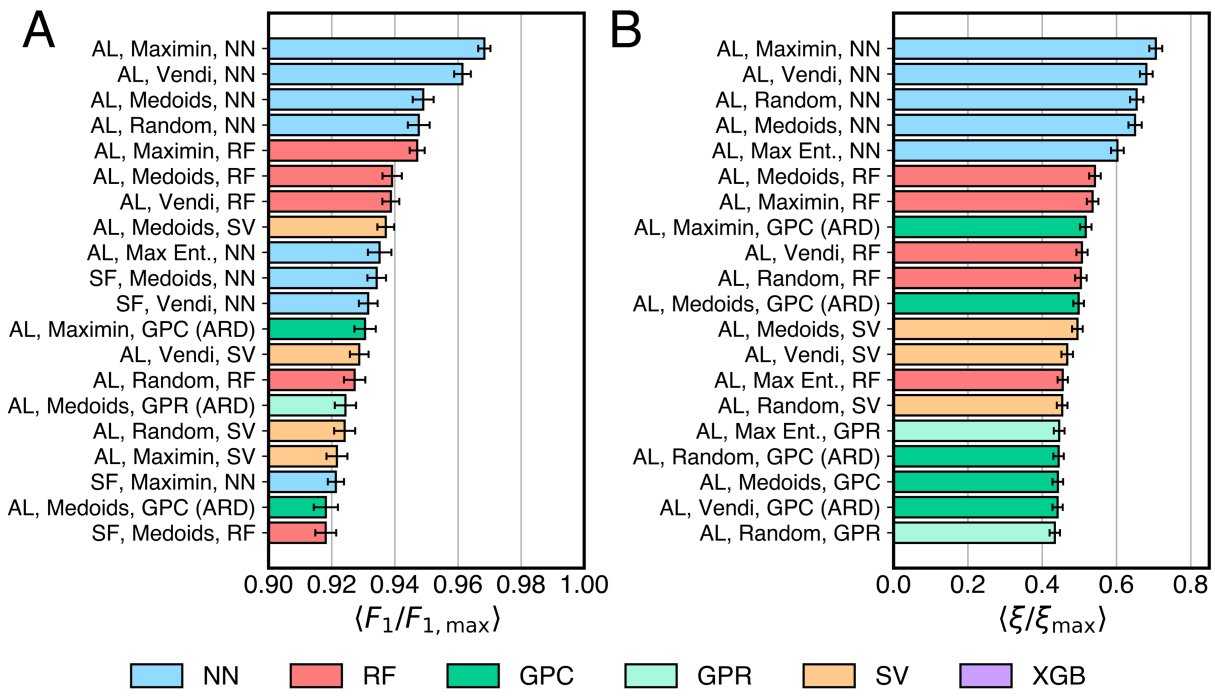

Figure S4: Summary of performances of the top-20 algorithms after five rounds of active learning on all tasks. Algorithm performance is measured by averaging the (A) relative Macro F_1 score and (B) relative ξ of each algorithm across all tasks, where "relative" denotes normalizing the metric by the performance of the top-performing algorithm on that task. Results are colored according to the model used by the specified algorithm. Error bars show the standard error.

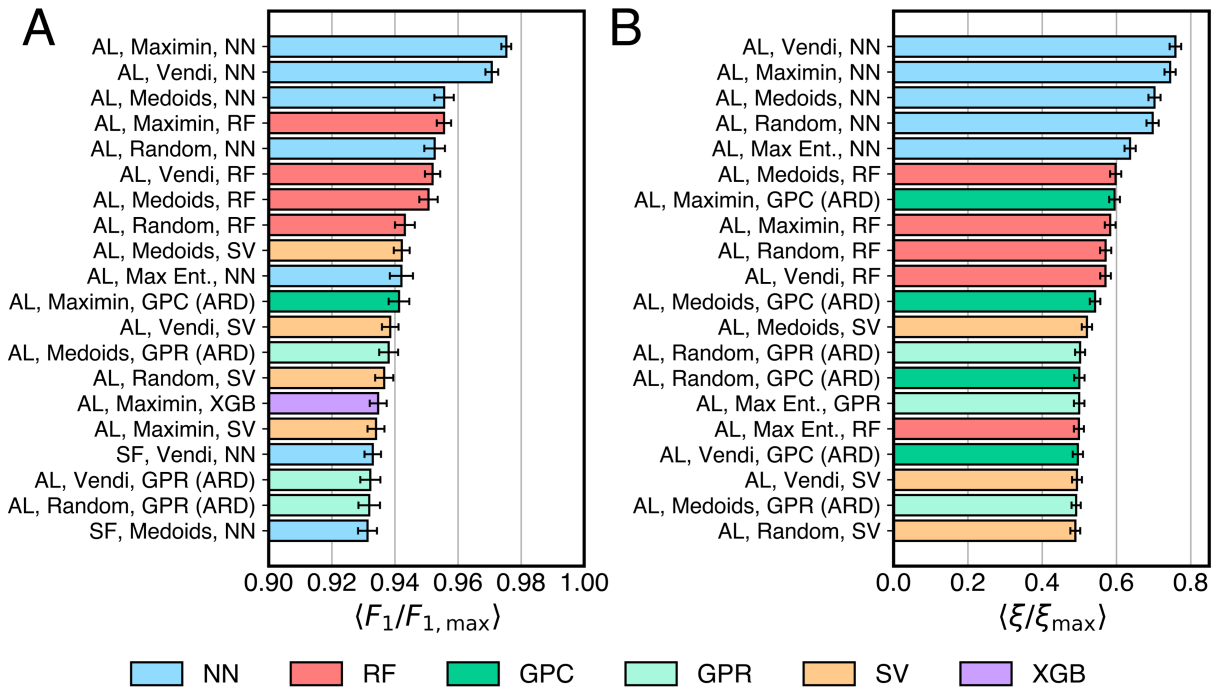

Figure S5: Summary of performances of the top-20 algorithms after seven rounds of active learning on all tasks. Algorithm performance is measured by averaging the (A) relative Macro F_1 score and (B) relative ξ of each algorithm across all tasks, where "relative" denotes normalizing the metric by the performance of the top-performing algorithm on that task. Results are colored according to the model used by the specified algorithm. Error bars show the standard error.

S3 Sensitivity of Active Learning vs. Space-Filling to Chosen Tasks

Figure 5 shows the frequency at which active learning algorithms out-perform space-filling algorithms with the same sampler, model, and seed for all tasks. In this section, we show the results for tasks which deviate from the trends shown in Figure 5. Specifically, we show the results for `princeton`, `tox21`, and `electro` tasks in Figures S6, S7, and S8, respectively.

Figure S6 shows that space-filling algorithms out-perform active learning algorithms on the `princeton` task. This is likely the case because of the complicated classification boundary present in the `princeton` task. Models that span the task domain are able to better characterize this complicated boundary, as opposed to active learning methods which tend to refine already discovered boundaries. Figures S7 and S8 show two tasks where there is not a monotonic increase in the frequency at which active learning algorithms out-perform space-filling algorithms with more rounds of active learning. We attribute this to the fact that the `tox21` and `electro` tasks are two of the most difficult classification tasks considered. When a task is very difficult, it is likely that both active learning and space-filling algorithms are struggling to accurately characterize the dataset. Therefore, the frequency at which active learning algorithms out-perform space-filling algorithms for these tasks is insignificant for identifying which scheme is more effective for accurately classifying points in the domain.

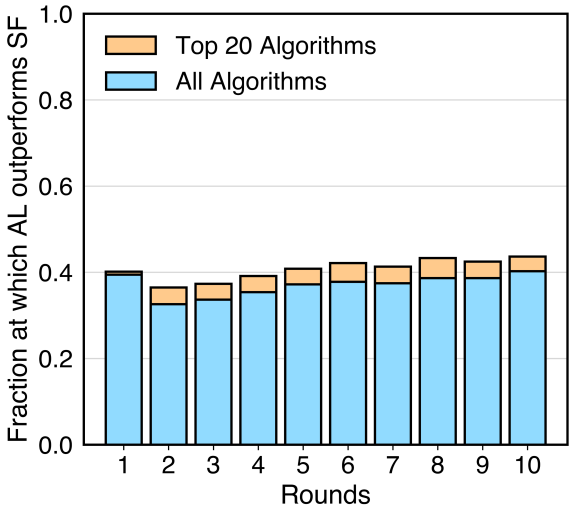

Figure S6: Fraction of sampler, model, and seed choices for which the active learning (AL) algorithm outperforms the equivalent space-filling (SF) algorithm based on the number of rounds of active learning. The fractions considering all sampler, model, and seed choices are shown in blue. The fractions considering only the top-20 sampler, model, and seed choices in Figure 3B include the blue and orange bars. Only resuts from the `princeton` task are shown.

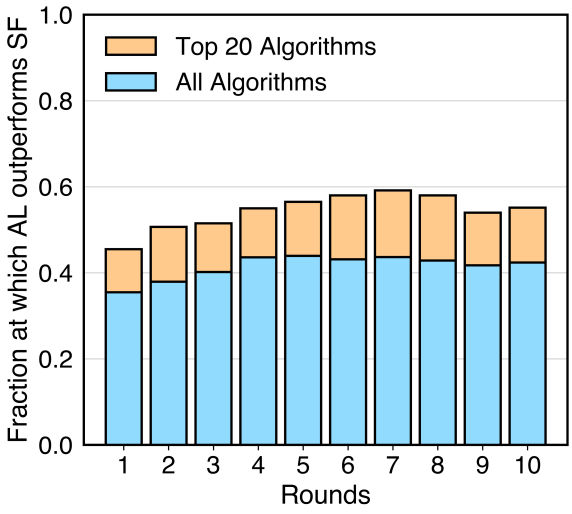

Figure S7: Fraction of sampler, model, and seed choices for which the active learning (AL) algorithm outperforms the equivalent space-filling (SF) algorithm based on the number of rounds of active learning. The fractions considering all sampler, model, and seed choices are shown in blue. The fractions considering only the top-20 sampler, model, and seed choices in Figure 3B include the blue and orange bars. Only resuts from the `tox21` task are shown.

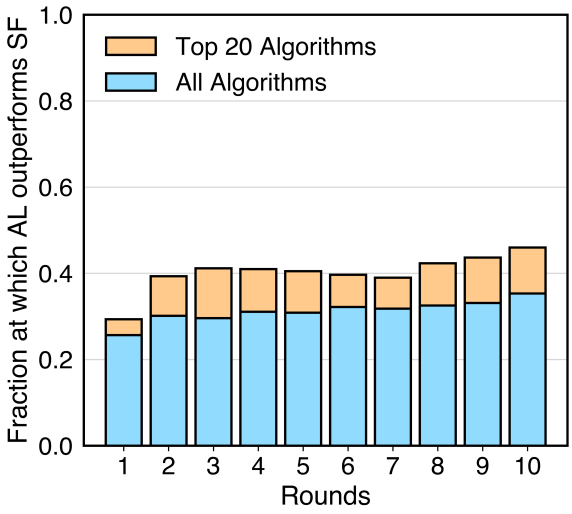

Figure S8: Fraction of sampler, model, and seed choices for which the active learning (AL) algorithm outperforms the equivalent space-filling (SF) algorithm based on the number of rounds of active learning. The fractions considering all sampler, model, and seed choices are shown in blue. The fractions considering only the top-20 sampler, model, and seed choices in Figure 3B include the blue and orange bars. Only resuts from the `electro` task are shown.

S4 Survey of Ensemble Strategies

We considered four approaches to combining the predictions and uncertainties of multiple models for developing data-efficient classification algorithms, inspired by those reported in Ref. [2]. First, we treated model choice as a hyperparameter. Here, the model with the highest cross-validation accuracy on the training set was the model selected to make predictions and compute uncertainties for the next round of active learning. Second, we considered simple averaging, where all considered models were trained on the training set and the combination of models whose average predictions obtained the highest cross-validation accuracy was used to compute uncertainties. Uncertainties were calculated by averaging the scaled uncertainties of the chosen models. Third, we considered stacking, which involves a linear model that takes as input individual model predictions and outputs a final prediction. In this way, stacking ensembles make predictions using learnable weights applied to the predictions of its component models. Uncertainties are calculated using a weighted sum of scaled uncertainties for each model. Finally, we consider ensembles built using arbitration, where predictions using separate models for each point. The model chosen for a given point is the one that is most certain in its prediction for that point, as judged by the lowest value of its scaled uncertainty. The uncertainty of that model for the specified point is taken to be the uncertainty of the ensemble at that point. Only random forests, neural networks, and anisotropic GPCs are considered by the ensemble scheme—these were shown to outperform ensembles built from random forests, neural networks, XGBs, and isotropic and anisotropic GPCs and GPRs.

Given the relatively poor performance of space-filling algorithms, only active learning algorithms were studied using ensemble schemes. Figure S9 shows the performances of ensemble-based active learning algorithms on all tasks. It is clear that ensemble-based algorithms which treat model choice as a hyperparameter are the top-performing algorithms. Interestingly, the maximin and medoids versions of the remaining ensemble methods perform better than those using alternative samplers. When considering only maximin and medoids samplers, it appears that ensemble-based algorithms which use averaging and stacking also perform well, but not as well as the methods which treat model choice as a hyperparameter.

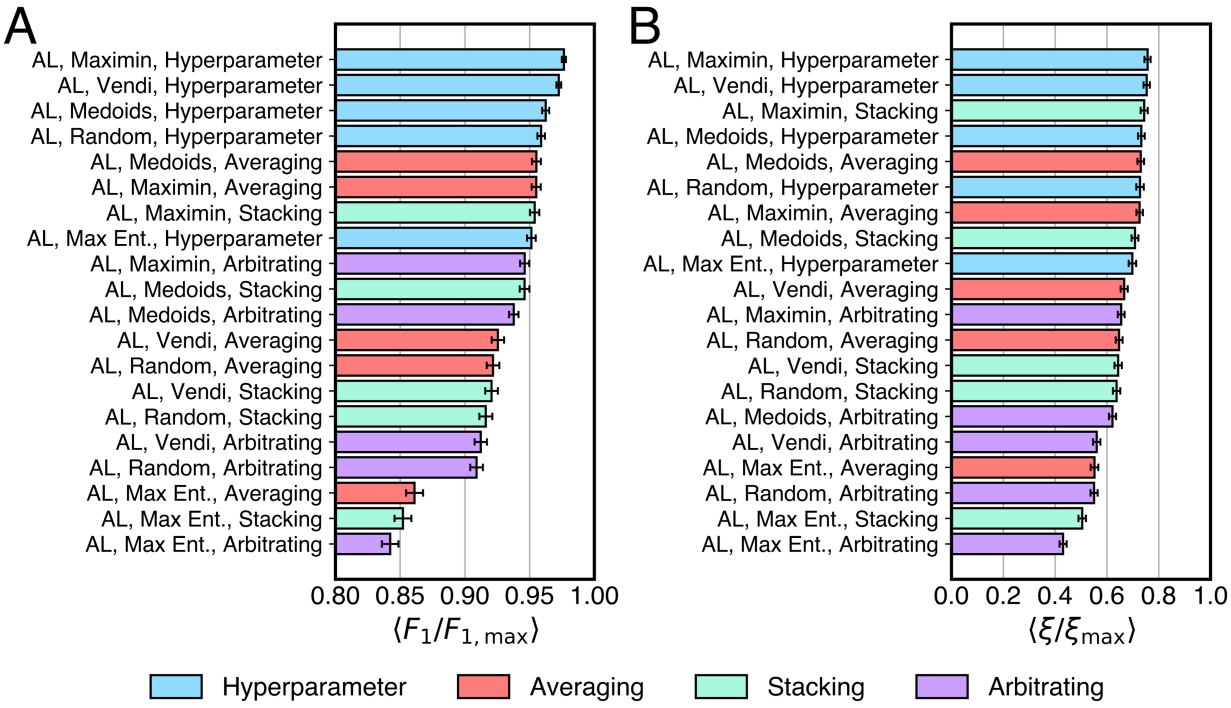

Figure S9: Summary of performances of the all ensemble-based active learning algorithms. Algorithm performance is measured by averaging the (A) relative Macro F_1 score and (B) relative ξ of each algorithm across all tasks, where "relative" denotes normalizing the metric by the performance of the top-performing algorithm on that task. Results are colored according to the ensemble scheme employed. Error bars show the standard error.

Table S1: Top-performing algorithms for each task. For "Type", AL denotes active learning and SF denotes space-filling. $\langle\xi\rangle/N$ is the average number of points measured by a naive algorithm to achieve the same performance as the top-performing algorithm ($\langle\xi\rangle$), divided by the number of points measured by the top-performing algorithm (N). This metric reflects how many more points must be measured by a naive approach to achieve the same accuracy as the specified strategy. For molecular datasets, the molecular representation of the top-performing algorithm is included with the model. Here, M(10), M(20), M(100), and M(All) refer to 10, 20, 100, and all available Mordred descriptors, respectively. FP refers to Morgan fingerprints of size 1024.

Task	Type	Sampler	Model	Macro F_1	$\langle\xi\rangle/N$
bace	AL	Random	RF-M(100)	0.774 ± 0.004	3.54
bear	AL	Max Entropy	NN	0.793 ± 0.006	2.79
clintox	AL	Maximin	RF-M(20)	0.791 ± 0.004	6.31
diblock	AL	Vendi	XGB	0.853 ± 0.004	2.97
electro	AL	Maximin	NN	0.960 ± 0.018	2.16
esol	AL	Medoids	RF-M(20)	0.908 ± 0.001	5.81
free	SF	Medoids	GPC-M(100)	0.912 ± 0.001	4.29
glotzer_pf	AL	Max Entropy	NN	0.999 ± 0.000	78.31
glotzer_xa	AL	Random	NN	0.998 ± 0.000	49.81
hiv	SF	Medoids	GPC-FP	0.694 ± 0.003	11.55
hplc	AL	Max Entropy	RF	0.849 ± 0.001	5.75
lipo	SF	Medoids	GPC (ARD)-M(20)	0.742 ± 0.006	5.03
muv	SF	Medoids	NN-M(100)	0.575 ± 0.004	3.577
oer	AL	Max Entropy	NN	0.871 ± 0.003	6.05
oxidation	AL	Maximin	NN	0.982 ± 0.002	8.43
perovskite	AL	Medoids	XGB	0.745 ± 0.008	4.17
polygel	AL	Max Entropy	RF	0.860 ± 0.002	19.48
polysol	AL	Medoids	RF	0.928 ± 0.003	7.34
princeton	AL	Maximin	GPR	0.942 ± 0.004	3.36
qm9_cv	AL	Random	SV-M(20)	0.936 ± 0.001	26.49
qm9_gap	AL	Maximin	NN-M(100)	0.804 ± 0.005	15.40
qm9_r2	AL	Vendi	RF-M(100)	0.925 ± 0.002	28.29
qm9_u0	AL	Random	RF-M(20)	0.999 ± 0.000	66.95
qm9_zpve	AL	Medoids	GPC (ARD)-M(20)	0.993 ± 0.000	55.89
robeson	AL	Maximin	SV-M(20)	0.920 ± 0.005	7.13
shower	AL	Max Entropy	GPC (ARD)	0.984 ± 0.001	11.55
toporg	AL	Maximin	NN	0.984 ± 0.001	11.55
tox21	AL	Max Entropy	RF-M(All)	0.654 ± 0.003	14.98
vdw	AL	Maximin	NN	0.977 ± 0.003	3.45
water_hp	AL	Maximin	NN	0.998 ± 0.001	5.42
water_lp	AL	Max Entropy	NN	0.995 ± 0.001	5.03

S5 Top-Performing Algorithms for Each Task

Table S1 shows the top-performing algorithm for each task. The vast majority of top-performing algorithms are active learning algorithms, with exceptions for free, hiv, lipo, and muv tasks. Maximin, medoids, and max entropy samplers are well-represented, while random and Vendi samplers are less common. In all cases where space-filling algorithms are top-performers, the medoids sampler is used. Models tend to include NNs, RFs, XGBs, and GPs. There are no top-performing algorithms based on SV, LP, or BKDE models. The scores of top-performing models vary from task to task, with some scores as high as 0.999 and some as low as 0.575. Values of $\langle\xi\rangle/N$, which reflects how many points must be measured by a naive algorithm to achieve the same accuracy as the top-performing algorithm, can be as high as 78.31. All values are greater than 2.00, indicating that the top-performing strategy is reducing the number of points measured by 50% for all tasks. For molecular tasks, the optimal choice of molecular representation is often Mordred descriptor vectors of length 20 or 100. There is only one instance each where including all Mordred descriptors or Morgan fingerprints is optimal, and these instances occur on the two most difficult classification tasks.

References

[1] Florian Häse, Matteo Aldeghi, Riley J. Hickman, Loïc M. Roch, and Alán Aspuru-Guzik. Gryffin: An algorithm for Bayesian optimization of categorical variables informed by expert knowledge. *Applied Physics Reviews*, 8(3):031406, July 2021.

[2] Pavel Brazdil, Jan N. Van Rijn, Carlos Soares, and Joaquin Vanschoren. *Metalearning: Applications to Automated Machine Learning and Data Mining*. Cognitive Technologies. Springer International Publishing, Cham, 2022.