Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn

Brent Komer*†, James Bergstra†, Chris Eliasmith†

Abstract—Hyperopt-sklearn is a new software project that provides automatic algorithm configuration of the Scikit-learn machine learning library. Following Auto-Weka, we take the view that the choice of classifier and even the choice of preprocessing module can be taken together to represent a *single large hyperparameter optimization problem*. We use Hyperopt to define a search space that encompasses many standard components (e.g. SVM, RF, KNN, PCA, TFIDF) and common patterns of composing them together. We demonstrate, using search algorithms in Hyperopt and standard benchmarking data sets (MNIST, 20-Newsgroups, Convex Shapes), that searching this space is practical and effective. In particular, we improve on best-known scores for the model space for both MNIST and Convex Shapes.

Index Terms—bayesian optimization, model selection, hyperparameter optimization, scikit-learn

Introduction

The size of data sets and the speed of computers have increased to the point where it is often easier to fit complex functions to data using statistical estimation techniques than it is to design them by hand. The fitting of such functions (training machine learning algorithms) remains a relatively arcane art, typically mastered in the course of a graduate degree and years of experience. Recently however, techniques for automatic algorithm configuration based on Regression Trees [Hut11], Gaussian Processes [Moc78], [Sno12], and density-estimation techniques [Ber11] have emerged as viable alternatives to hand-tuning by domain specialists.

Hyperparameter optimization of machine learning systems was first applied to neural networks, where the number of parameters can be overwhelming. For example, [Ber11] tuned Deep Belief Networks (DBNs) with up to 32 hyperparameters, and [Ber13a] showed that similar methods could search a 238-dimensional configuration space describing multi-layer convolutional networks (convnets) for image classification.

Relative to DBNs and convnets, algorithms such as Support Vector Machines (SVMs) and Random Forests (RFs) have a small-enough number of hyperparameters that manual tuning and grid or random search provides satisfactory results. Taking a step back though, there is often no particular reason to use either an SVM or an RF when they are both computationally viable. A model-agnostic practitioner may simply prefer to go with the one that provides greater accuracy. In this light, *the choice of classifier can be seen as hyperparameter* alongside the C-value in the SVM and the max-tree-depth of the RF. Indeed the choice and configuration of *preprocessing* components may likewise be seen as part of the model selection / hyperparameter optimization problem.

The Auto-Weka project [Tho13] was the first to show that an entire library of machine learning approaches (Weka [Hal09]) can be searched within the scope of a single run of hyperparameter tuning. However, Weka is a GPL-licensed Java library, and was not written with scalability in mind, so we feel there is a need for alternatives to Auto-Weka. Scikit-learn [Ped11] is another library of machine learning algorithms. Is written in Python (with many modules in C for greater speed), and is BSD-licensed. Scikit-learn is widely used in the scientific Python community and supports many machine learning application areas.

With this paper we introduce Hyperopt-Sklearn: a project that brings the benefits of automatic algorithm configuration to users of Python and scikit-learn. Hyperopt-Sklearn uses Hyperopt [Ber13b] to describe a search space over possible configurations of Scikit-Learn components, including preprocessing and classification modules. Section 2 describes our configuration space of 6 classifiers and 5 preprocessing modules that encompasses a strong set of classification systems for dense and sparse feature classification (of images and text). Section 3 presents experimental evidence that search over this space is viable, meaningful, and effective. Section 4 presents a discussion of the results, and directions for future work.

Background: Hyperopt for Optimization

The Hyperopt library [Ber13b] offers optimization algorithms for search spaces that arise in algorithm configuration. These spaces are characterized by a variety of types of variables (continuous, ordinal, categorical), different sensitivity profiles (e.g. uniform vs. log scaling), and conditional structure (when there is a choice between two classifiers, the parameters of one classifier are irrelevant when the other classifier is chosen). To use Hyperopt, a user must define/choose three things:

1. a search domain,
2. an objective function,

* *Corresponding author: bjkomer@uwaterloo.ca*
† *Centre for Theoretical Neuroscience, University of Waterloo*

3. an optimization algorithm.

The search domain is specified via random variables, whose distributions should be chosen so that the most promising combinations have high prior probability. The search domain can include Python operators and functions that combine random variables into more convenient data structures for the objective function. The objective function maps a joint sampling of these random variables to a scalar-valued score that the optimization algorithm will try to *minimize*. Having chosen a search domain, an objective function, and an optimization algorithm, Hyperopt's `fmin` function carries out the optimization, and stores results of the search to a database (e.g. either a simple Python list or a MongoDB instance). The *fmin* call carries out the simple analysis of finding the best-performing configuration, and returns that to the caller. The *fmin* call can use multiple workers when using the MongoDB backend, to implement parallel model selection on a compute cluster.

Scikit-Learn Model Selection as a Search Problem

Model selection is the process of estimating which machine learning model performs best from among a possibly infinite set of possibilities. As an optimization problem, the search domain is the set of valid assignments to the configuration parameters (hyperparameters) of the machine learning model, and the objective function is typically cross-validation, the negative degree of success on held-out examples. Practitioners usually address this optimization by hand, by grid search, or by random search. In this paper we discuss solving it with the Hyperopt optimization library. The basic approach is to set up a search space with random variable hyperparameters, use scikit-learn to implement the objective function that performs model training and model validation, and use Hyperopt to optimize the hyperparamters.

Scikit-learn includes many algorithms for classification (classifiers), as well as many algorithms for preprocessing data into the vectors expected by classification algorithms. Classifiers include for example, K-Neighbors, SVM, and RF algorithms. Preprocessing algorithms include things like component-wise Z-scaling (Normalizer) and Principle Components Analysis (PCA). A full classification algorithm typically includes a series of preprocessing steps followed by a classifier. For this reason, scikit-learn provides a *pipeline* data structure to represent and use a sequence of preprocessing steps and a classifier as if they were just one component (typically with an API similar to the classifier). Although hyperopt-sklearn does not formally use scikit-learn's pipeline object, it provides related functionality. Hyperopt-sklearn provides a parameterization of a *search space* over pipelines, that is, of sequences of preprocessing steps and classifiers.

The configuration space we provide includes six preprocessing algorithms and seven classification algorithms. The full search space is illustrated in Figure 1. The preprocessing algorithms were (by class name, followed by n. hyperparameters + n. unused hyperparameters): `PCA(2)`, `StandardScaler(2)`, `MinMaxScaler(1)`, `Normalizer(1)`, `None`, and `TF-IDF(0+9)`. The first four preprocessing algorithms were for dense features. PCA performed whitening or non-whitening principle components analysis. The `StandardScaler`, `MinMaxScaler`, and `Normalizer` did various feature-wise affine transforms to map numeric input features onto values near 0 and with roughly unit variance. The `TF-IDF` preprocessing module performed feature extraction from text data. The classification algorithms were (by class name (used + unused hyperparameters)): `SVC(23)`, `KNN(4+5)`, `RandomForest(8)` , `ExtraTrees(8)` , `SGD(8+4)` , and `MultinomialNB(2)` . The `SVC` module is a fork of LibSVM, and our wrapper has 23 hyperparameters because we treated each possible kernel as a different classifier, with its own set of hyperparameters: Linear(4), RBF(5), Polynomial(7), and Sigmoid(6). In total, our parameterization has 65 hyperparameters: 6 for preprocessing and 53 for classification. The search space includes 15 boolean variables, 14 categorical, 17 discrete, and 19 real-valued variables.

Although the total number of hyperparameters is large, the number of *active* hyperparameters describing any one model is much smaller: a model consisting of `PCA` and a `RandomForest` for example, would have only 12 active hyperparameters (1 for the choice of preprocessing, 2 internal to PCA, 1 for the choice of classifier and 8 internal to the RF). Hyperopt description language allows us to differentiate between *conditional* hyperparameters (which must always be assigned) and *non-conditional* hyperparameters (which may remain unassigned when they would be unused). We make use of this mechanism extensively so that Hyperopt's search algorithms do not waste time learning by trial and error that e.g. RF hyperparameters have no effect on SVM performance. Even internally within classifiers, there are instances of conditional parameters: `KNN` has conditional parameters depending on the distance metric, and `LinearSVC` has 3 binary parameters (`loss` , `penalty` , and `dual`) that admit only 4 valid joint assignments. We also included a blacklist of (preprocessing, classifier) pairs that did not work together, e.g. PCA and MinMaxScaler were incompatible with MultinomialNB, TF-IDF could only be used for text data, and the tree-based classifiers were not compatible with the sparse features produced by the TF-IDF preprocessor. Allowing for a 10-way discretization of real-valued hyperparameters, and taking these conditional hyperparameters into account, a grid search of our search space would still require an infeasible number of evalutions (on the order of 10^{12}).

Finally, the search space becomes an optimization problem when we also define a scalar-valued search *objective*. Hyperopt-sklearn uses scikit-learn's *score* method on *validation data* to define the search criterion. For classifiers, this is the so-called "Zero-One Loss": the number of correct label predictions among data that has been withheld from the data set used for training (and also from the data used for testing *after* the model selection search process).

Example Usage

Following Scikit-learn's convention, hyperopt-sklearn provides an `Estimator` class with a `fit` method and a `predict`

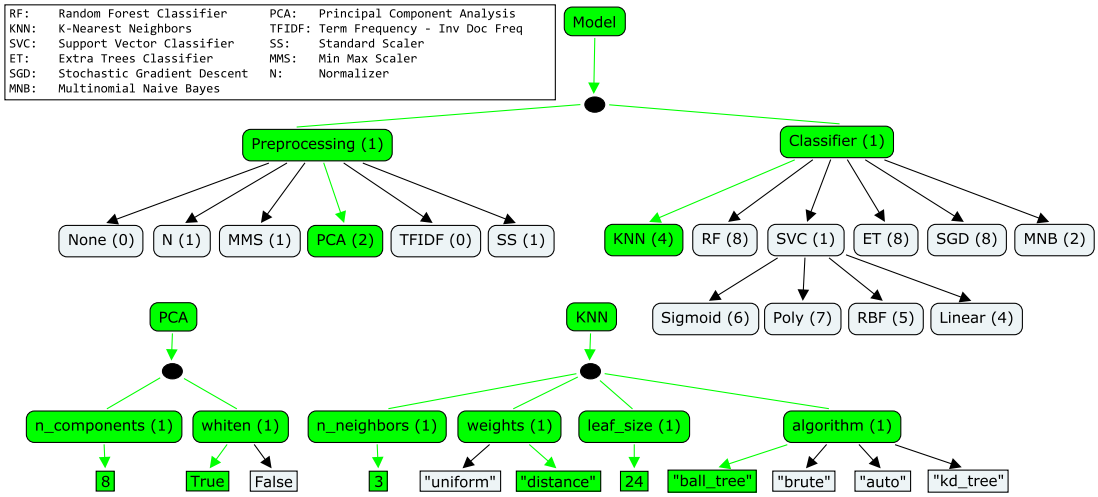

Fig. 1: *Hyperopt-sklearn's full search space ("Any Classifier") consists of a (preprocessing, classsifier) pair. There are 6 possible preprocessing modules and 6 possible classifiers. Choosing a model within this configuration space means choosing paths in an ancestral sampling process. The highlighted green edges and nodes represent a (PCA, K-Nearest Neighbor) model. The number of active hyperparameters in a model is the sum of parenthetical numbers in the selected boxes. For the PCA+KNN combination, 7 hyperparameters are activated.*

method. The `fit` method of this class performs hyperparameter optimization, and after it has completed, the `predict` method applies the best model to test data. Each evaluation during optimization performs training on a large fraction of the training set, estimates test set accuracy on a validation set, and returns that validation set score to the optimizer. At the end of search, the best configuration is retrained on the whole data set to produce the classifier that handles subsequent `predict` calls.

One of the important goals of hyperopt-sklearn is that it is easy to learn and to use. To facilitate this, the syntax for fitting a classifier to data and making predictions is very similar to scikit-learn. Here is the simplest example of using this software.

```
from hpsklearn import HyperoptEstimator
# Load data ({train,test}_{data,label})
# Create the estimator object
estim = HyperoptEstimator()
# Search the space of classifiers and preprocessing
# steps and their respective hyperparameters in
# scikit-learn to fit a model to the data
estim.fit(train_data, train_label)
# Make a prediction using the optimized model
prediction = estim.predict(unknown_data)
# Report the accuracy of the classifier
# on a given set of data
score = estim.score(test_data, test_label)
# Return instances of the classifier and
# preprocessing steps
model = estim.best_model()
```

The `HyperoptEstimator` object contains the information of what space to search as well as how to search it. It can be configured to use a variety of hyperparameter search algorithms and also supports using a combination of algorithms. Any algorithm that supports the same interface as the algorithms in hyperopt can be used here. This is also where you, the user, can specify the maximum number of function evaluations you would like to be run as well as a timeout (in seconds) for each run.

```
from hpsklearn import HyperoptEstimator
from hyperopt import tpe
estim = HyperoptEstimator(algo=tpe.suggest,
                          max_evals=150,
                          trial_timeout=60)
```

Each search algorithm can bring its own bias to the search space, and it may not be clear that one particular strategy is the best in all cases. Sometimes it can be helpful to use a mixture of search algorithms.

```
from hpsklearn import HyperoptEstimator
from hyperopt import anneal, rand, tpe, mix
# define an algorithm that searches randomly 5% of
# the time, uses TPE 75% of the time, and uses
# annealing 20% of the time
mix_algo = partial(mix.suggest, p_suggest=[
        (0.05, rand.suggest),
        (0.75, tpe.suggest),
        (0.20, anneal.suggest)])
estim = HyperoptEstimator(algo=mix_algo,
                          max_evals=150,
                          trial_timeout=60)
```

Searching effectively over the entire space of classifiers available in scikit-learn can use a lot of time and computational resources. Sometimes you might have a particular subspace of models that they are more interested in. With hyperopt-sklearn it is possible to specify a more narrow search space to allow it to be be explored in greater depth.

```
from hpsklearn import HyperoptEstimator, svc
# limit the search to only models a SVC
estim = HyperoptEstimator(classifier=svc('my_svc'))
```

Combinations of different spaces can also be used.

```
from hpsklearn import HyperoptEstimator, svc, knn, \
from hyperopt import hp
# restrict the space to contain only random forest,
# k-nearest neighbors, and SVC models.
clf = hp.choice('my_name',
      [random_forest('my_name.random_forest'),
       svc('my_name.svc'),
       knn('my_name.knn')])
estim = HyperoptEstimator(classifier=clf)
```

The support vector machine provided by scikit-learn has a number of different kernels that can be used (linear, rbf, poly,

sigmoid). Changing the kernel can have a large effect on the performance of the model, and each kernel has its own unique hyperparameters. To account for this, hyperopt-sklearn treats each kernel choice as a unique model in the search space. If you already know which kernel works best for your data, or you are just interested in exploring models with a particular kernel, you may specify it directly rather than going through the `svc`.

```
from hpsklearn import HyperoptEstimator, svc_rbf
estim = HyperoptEstimator(
          classifier=svc_rbf('my_svc'))
```

It is also possible to specify which kernels you are interested in by passing a list to the `svc`.

```
from hpsklearn import HyperoptEstimator, svc
estim = HyperoptEstimator(
      classifier=svc('my_svc',
                     kernels=['linear',
                              'sigmoid']))
```

In a similar manner to classifiers, the space of preprocessing modules can be fine tuned. Multiple successive stages of preprocessing can be specified by putting them in a list. An empty list means that no preprocessing will be done on the data.

```
from hpsklearn import HyperoptEstimator, pca
estim = HyperoptEstimator(
      preprocessing=[pca('my_pca')])
```

Combinations of different spaces can be used here as well.

```
from hpsklearn import HyperoptEstimator, tfidf, pca
from hyperopt import hp
preproc = hp.choice('my_name',
      [[pca('my_name.pca')],
       [pca('my_name.pca'), normalizer('my_name.norm')]
       [standard_scaler('my_name.std_scaler')],
       []])
estim = HyperoptEstimator( preprocessing=preproc )
```

Some types of preprocessing will only work on specific types of data. For example, the TfidfVectorizer that scikit-learn provides is designed to work with text data and would not be appropriate for other types of data. To address this, hyperopt-sklearn comes with a few pre-defined spaces of classifiers and preprocessing tailored to specific data types.

```
from hpsklearn import HyperoptEstimator, \
                      any_sparse_classifier, \
                      any_text_preprocessing
from hyperopt import tpe
estim = HyperoptEstimator(
      algo=tpe.suggest,
      classifier=any_sparse_classifier('my_clf')
      preprocessing=any_text_preprocessing('my_pp')
      max_evals=200,
      trial_timeout=60 )
```

So far in all of these examples, every hyperparameter available to the model is being searched over. It is also possible for you to specify the values of specific hyperparameters, and those parameters will remain constant during the search. This could be useful, for example, if you knew you wanted to use whitened PCA data and a degree-3 polynomial kernel SVM.

```
from hpsklearn import HyperoptEstimator, pca, svc_poly
estim = HyperoptEstimator(
          preprocessing=[pca('my_pca', whiten=True)],
          classifier=svc_poly('my_poly', degree=3))
```

It is also possible to specify ranges of individual parameters. This is done using the standard hyperopt syntax. These will override the defaults defined within hyperopt-sklearn.

```
from hpsklearn import HyperoptEstimator, pca, sgd
from hyperopt import hp
import numpy as np
sgd_loss = hp.pchoice('loss',
                      [(0.50, 'hinge'),
                       (0.25, 'log'),
                       (0.25, 'huber')])
sgd_penalty = hp.choice('penalty',
                        ['l2', 'elasticnet'])
sgd_alpha = hp.loguniform('alpha',
                          low=np.log(1e-5),
                          high=np.log(1) )
estim = HyperoptEstimator(
          classifier=sgd('my_sgd',
                         loss=sgd_loss,
                         penalty=sgd_penalty,
                         alpha=sgd_alpha) )
```

All of the components available to the user can be found in the `components.py` file. A complete working example of using hyperopt-sklearn to find a model for the 20 newsgroups data set is shown below.

```
from hpsklearn import HyperoptEstimator, tfidf, \
                      any_sparse_classifier
from sklearn.datasets import fetch_20newsgroups
from hyperopt import tpe
import numpy as np
# Download data and split training and test sets
train = fetch_20newsgroups(subset='train')
test = fetch_20newsgroups(subset='test')
X_train = train.data
y_train = train.target
X_test = test.data
y_test = test.target
estim = HyperoptEstimator(
          classifier=any_sparse_classifier('clf'),
          preprocessing=[tfidf('tfidf')],
          algo=tpe.suggest,
          trial_timeout=180)
estim.fit(X_train, y_train)
print(estim.score(X_test, y_test))
print(estim.best_model())
```

Experiments

We conducted experiments on three data sets to establish that hyperopt-sklearn can find accurate models on a range of data sets in a reasonable amount of time. Results were collected on three data sets: MNIST, 20-Newsgroups, and Convex Shapes. MNIST is a well-known data set of 70K 28x28 greyscale images of hand-drawn digits [Lec98]. 20-Newsgroups is a 20-way classification data set of 20K newsgroup messages ([Mit96] , we did not remove the headers for our experiments). Convex Shapes is a binary classification task of distinguishing pictures of convex white-colored regions in small (32x32) black-and-white images [Lar07].

Figure 2 shows that there was no penalty for searching broadly. We performed optimization runs of up to 300 function evaluations searching the entire space, and compared the quality of solution with specialized searches of specific classifier types (including best known classifiers).

Figure 3 shows that search could find different, good models. This figure was constructed by running hyperopt-sklearn with different initial conditions (number of evaluations,

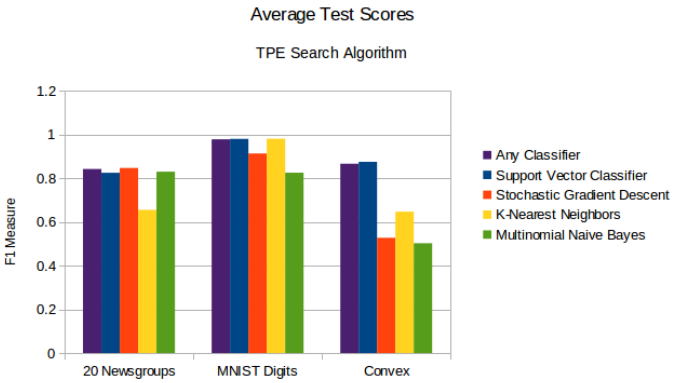

Fig. 2: *For each data set, searching the full configuration space ("Any Classifier") delivered performance approximately on par with a search that was restricted to the best classifier type. (Best viewed in color.)*

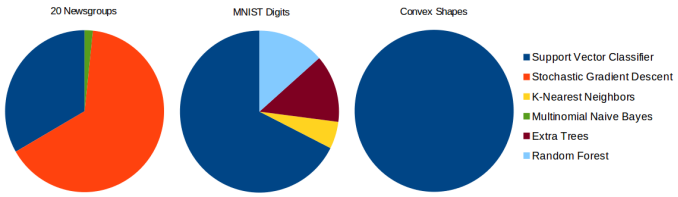

Fig. 3: *Looking at the best models from all optimization runs performed on the full search space (using different initial conditions, and different optimization algorithms) we see that different data sets are handled best by different classifiers. SVC was the only classifier ever chosen as the best model for Convex Shapes, and was often found to be best on MNIST and 20 Newsgroups, however the best SVC parameters were very different across data sets.*

choice of optimization algorithm, and random number seed) and keeping track of what final model was chosen after each run. Although support vector machines were always among the best, the parameters of best SVMs looked very different across data sets. For example, on the image data sets (MNIST and Convex) the SVMs chosen never had a sigmoid or linear kernel, while on 20 newsgroups the linear and sigmoid kernel were often best.

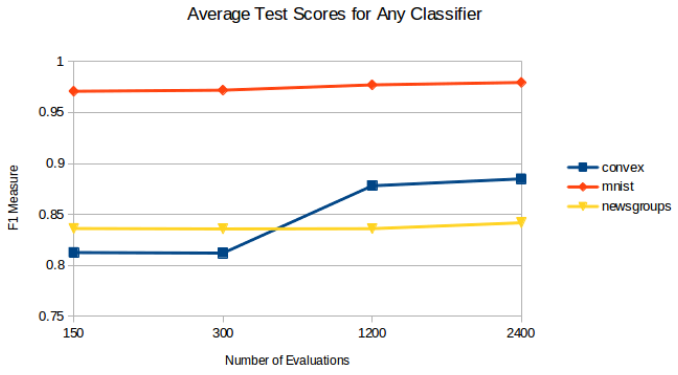

Fig. 4: *Using Hyperopt's Anneal search algorithm, increasing the number of function evaluations from 150 to 2400 lead to a modest improvement in accuracy on 20 Newsgroups and MNIST, and a more dramatic improvement on Convex Shapes. We capped evaluations to 5 minutes each so 300 evaluations took between 12 and 24 hours of wall time.*

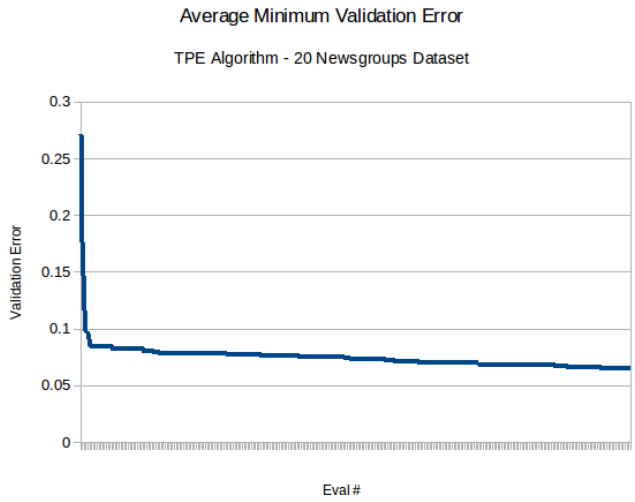

Fig. 5: *Right: TPE makes gradual progress on 20 Newsgroups over 300 iterations and gives no indication of convergence.*

Discussion and Future Work

Table 1 lists the test set scores of the best models found by cross-validation, as well as some points of reference from previous work. Hyperopt-sklearn's scores are relatively good on each data set, indicating that with hyperopt-sklearn's parameterization, Hyperopt's optimization algorithms are competitive with human experts.

The model with the best performance on the MNIST Digits data set uses deep artificial neural networks. Small receptive fields of convolutional winner-take-all neurons build up the large network. Each neural column becomes an expert on inputs preprocessed in different ways, and the average prediction of 35 deep neural columns to come up with a single final prediction [Cir12]. This model is much more advanced than those available in scikit-learn. The previously best known model in the scikit-learn search space is a radial-basis SVM on centered data that scores 98.6%, and hyperopt-sklearn matches that performance [MNIST].

The CFC model that performed quite well on the 20 newsgroups document classification data set is a Class-Feature-Centroid classifier. Centroid approaches are typically inferior to an SVM, due to the centroids found during training being far from the optimal location. The CFC method reported here uses a centroid built from the inter-class term index and the inner-class term index. It uses a novel combination of these indices along with a denormalized cosine measure to calculate the similarity score between the centroid and a text vector [Gua09]. This style of model is not currently implemented in hyperopt-sklearn, and our experiments suggest that existing hyperopt-sklearn components cannot be assembled to match its level of performance. Perhaps when it is implemented, Hyperopt may find a set of parameters that provides even greater classification accuracy.

On the Convex Shapes data set, our Hyperopt-sklearn experiments revealed a more accurate model than was previously believed to exist in any search space, let alone a search space of such standard components. This result underscores

MNIST		20 Newsgroups		Convex Shapes	
Approach	Accuracy	Approach	F-Score	Approach	Accuracy
Committee of convnets	99.8%	CFC	0.928	**hyperopt-sklearn**	**88.7%**
hyperopt-sklearn	**98.7%**	**hyperopt-sklearn**	**0.856**	hp-dbnet	84.6%
libSVM grid search	98.6%	SVMTorch	0.848	dbn-3	81.4%
Boosted trees	98.5%	LibSVM	0.843		

TABLE 1: *Hyperopt-sklearn scores relative to selections from literature on the three data sets used in our experiments. On MNIST, hyperopt-sklearn is one of the best-scoring methods that does not use image-specific domain knowledge (these scores and others may be found at http://yann.lecun.com/exdb/mnist/). On 20 Newsgroups, hyperopt-sklearn is competitive with similar approaches from the literature (scores taken from [Gua09]). In the 20 Newsgroups data set, the score reported for hyperopt-sklearn is the weighted-average F1 score provided by sklearn. The other approaches shown here use the macro-average F1 score. On Convex Shapes, hyperopt-sklearn outperforms previous automatic algorithm configuration approaches [Egg13] and manual tuning [Lar07] .*

the difficulty and importance of hyperparameter search.

Hyperopt-sklearn provides many opportunities for future work: more classifiers and preprocessing modules could be included in the search space, and there are more ways to combine even the existing components. Other types of data require different preprocessing, and other prediction problems exist beyond classification. In expanding the search space, care must be taken to ensure that the benefits of new models outweigh the greater difficulty of searching a larger space. There are some parameters that scikit-learn exposes that are more implementation details than actual hyperparameters that affect the fit (such as `algorithm` and `leaf_size` in the KNN model). Care should be taken to identify these parameters in each model and they may need to be treated differently during exploration.

It is possible for a user to add their own classifier to the search space as long as it fits the scikit-learn interface. This currently requires some understanding of how hyperopt-sklearn's code is structured and it would be nice to improve the support for this so minimal effort is required by the user. We also plan to allow the user to specify alternate scoring methods besides just accuracy and F-measure, as there can be cases where these are not best suited to the particular problem.

We have shown here that Hyperopt's random search, annealing search, and TPE algorithms make Hyperopt-sklearn viable, but the slow convergence in e.g. Figure 4 and 5 suggests that other optimization algorithms might be more call-efficient. The development of Bayesian optimization algorithms is an active research area, and we look forward to looking at how other search algorithms interact with hyperopt-sklearn's search spaces. Hyperparameter optimization opens up a new art of matching the parameterization of search spaces to the strengths of search algorithms.

Computational wall time spent on search is of great practical importance, and hyperopt-sklearn currently spends a significant amount of time evaluating points that are un-promising. Techniques for recognizing bad performers early could speed up search enormously [Swe14], [Dom14]. Relatedly, hyperopt-sklearn currently lacks support for K-fold cross-validation. In that setting, it will be crucial to follow SMAC in the use of racing algorithms to skip un-necessary folds.

Conclusions

We have introduced Hyperopt-sklearn, a Python package for automatic algorithm configuration of standard machine learning algorithms provided by Scikit-Learn. Hyperopt-sklearn provides a unified view of 6 possible preprocessing modules and 6 possible classifiers, yet with the help of Hyperopt's optimization functions it is able to both rival and surpass human experts in algorithm configuration. We hope that it provides practitioners with a useful tool for the development of machine learning systems, and automatic machine learning researchers with benchmarks for future work in algorithm configuration.

Acknowledgements

This research was supported by the NSERC Banting Fellowship program, the NSERC Engage Program and by D-Wave Systems. Thanks also to Hristijan Bogoevski for early drafts of a hyperopt-to-scikit-learn bridge.

References

[Ber11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. *Algorithms for hyper-parameter optimization*, NIPS, 24:2546–2554, 2011.

[Ber13a] J. Bergstra, D. Yamins, and D. D. Cox. *Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures*, In Proc. ICML, 2013a.

[Ber13b] J. Bergstra, D. Yamins, and D. D. Cox. *Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms*, SciPy'13, 2013b.

[Cir12] D. Ciresan, U. Meier, and J. Schmidhuber. *Multi-column Deep Neural Networks for Image Classification*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3642-3649. 2012.

[Dom14] T. Domhan, T. Springenberg, F. Hutter. *Extrapolating Learning Curves of Deep Neural Networks*, ICML AutoML Workshop, 2014.

[Egg13] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. *Towards an empirical foundation for assessing bayesian optimization of hyperparameters*, NIPS workshop on Bayesian Optimization in Theory and Practice, 2013.

[Gua09] H. Guan, J. Zhou, and M. Guo. *A class-feature-centroid classifier for text categorization*, Proceedings of the 18th international conference on World wide web, 201-210. ACM, 2009.

[Hal09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. *The weka data mining software: an update*, ACM SIGKDD explorations newsletter, 11(1):10-18, 2009.

[Hut11] F. Hutter, H. Hoos, and K. Leyton-Brown. *Sequential model-based optimization for general algorithm configuration*, LION-5, 2011. Extended version as UBC Tech report TR-2010-10.

[Lar07] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. *An empirical evaluation of deep architectures on problems with many factors of variation*, ICML, 473-480, 2007.

[Lec98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[Mit96] T. Mitchell. *20 newsgroups data set*, http://qwone.com/jason/20Newsgroups/, 1996.

[Moc78] J. Mockus, V. Tiesis, and A. Zilinskas. *The application of Bayesian methods for seeking the extremum*, L.C.W. Dixon and G.P. Szego, editors, Towards Global Optimization, volume 2, pages 117–129. North Holland, New York, 1978.

[MNIST] The MNIST Database of handwritten digits: http://yann.lecun.com/exdb/mnist/

[Ped11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, 12:2825–2830, 2011.

[Sno12] J. Snoek, H. Larochelle, and R. P. Adams. *Practical Bayesian optimization of machine learning algorithms*, Neural Information Processing Systems, 2012.

[Swe14] K. Swersky, J. Snoek, R.P. Adams. *Freeze-Thaw Bayesian Optimization*, arXiv:1406.3896, 2014.

[Tho13] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. *Auto-WEKA: Automated selection and hyper-parameter optimization of classification algorithms*, KDD 847-855, 2013.