Model Grammars

Bill McDowell
wmcdowel@stanford.edu

Stephan Eismann
seismann@stanford.edu

Abstract

We desire methods that navigate a space of feature sets and parameter architectures to discover models that are both human-interpretable and predictive. With this in mind, we introduce "feature grammars" which consist of production rules that construct complex features from simpler features, and "model grammars" that build complex neural network architectures from simpler architectures. Starting from a linear architecture over a small feature set, we interleave the application of the grammar rules with gradient-based parameter updates to explore a space of features and interpretable architectures, while simultaneously maximizing likelihood. We test our methods, and characterize their behavior on synthetic data.

1 Introduction

Many machine learning tasks require models to determine the relationships between logically complex features of an object. The space of such features may be too large to explore exhaustively, and so models must use an efficient method to navigate the space in search of features relevant to the task. For example, tasks in computational chemistry require models to learn the relationships between a molecule's atomic structure and its quantum mechanical properties [13]. The structure of a molecule determines an enormous space of logically complex, substructural features for a model to consider in predicting molecular properties. In this paper, we consider models that learn to perform such tasks over large feature spaces, defined inductively through "feature grammars" and "model grammars".

A "feature grammar" consists of a set of rules for constructing increasingly complex features from simpler features, and it implicitly defines a discrete space of features where two features are neighbors if one can be constructed from another by the application of deterministic feature transformation rules. Our models use such rules to navigate a feature space in search of relevant features under the assumption that the grammatical relationship between two features in the space encodes some information about their relevance for a task. Similarly, a "model grammar" extends the notion of a feature grammar to include rules that transform a neural network architecture during training. We experiment with models that use these rules to discover complex, human-interpretable network architectures.

Our current project extends our work from previous courses where we tested linear feature grammar models on synthetic first-order logic data sets, and also applied them to tasks of predicting properties of small organic molecules [15, 16]. In this project, we extend this past work by reproducing our feature grammar models and experiments using the PyTorch deep learning library [17]. Use of this library allows us to expand our "feature grammars" into "model grammars" that extend the automated search of the input feature space to a wider search over neural model architectures.

2 Methods[1]

We re-implement our existing linear feature grammar models [15, 16] using the PyTorch neural network library [17] (described in Section 2.1 below). This library allows us to implement deep tree-structured neural architectures searched by a model grammar (described in Section 2.2 below).

2.1 Linear Models with Feature Grammars

Our linear feature grammar models embed feature grammars into stochastic optimization algorithms for linear and logistic regression. When training traditional linear and logistic regression models, we are given labeled data instances

[1] We delay our discussion of related work and data until Sections 3.and 4

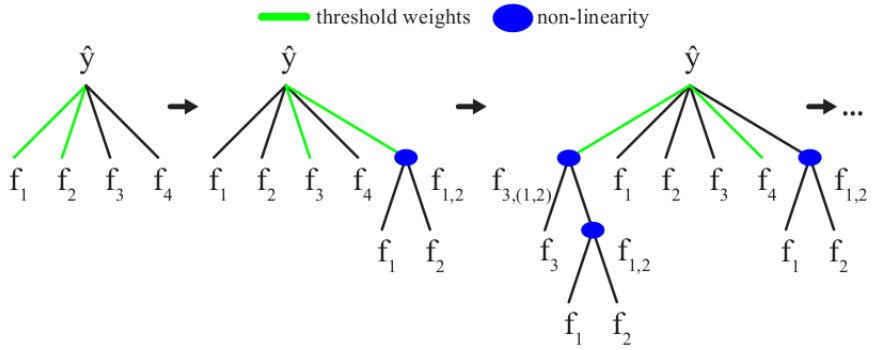

Figure 1: Tree architectures constructed through a model grammar. At initialization, the architecture is linear over the input features, but at the first step, the tree grammar rule is applied to high weighted features $\mathbf{f}_1$ and $\mathbf{f}_2$ (in green), to construct a new feature $\mathbf{f}_{1,2}$ that is a non-linear combination of the input two input features. The second step constructs a feature of more depth from $\mathbf{f}_{1,2}$ and $\mathbf{f}_3$. The algorithm continues to construct deeper features over further iterations.

$(\mathbf{x}, l) \in X \times \mathbb{R}$ with input vectors given by a feature function $\mathbf{f} : X \to \mathbb{R}^m$ [6]. Furthermore, for linear regression, we assume a label l is distributed, conditioned on input $\mathbf{x}$ with parameters $(\mathbf{w}, \sigma)$, according to $(l \mid x; \mathbf{w}, \sigma, \mathbf{f}) \sim \mathcal{N}(\mathbf{w}^\top \mathbf{f}(\mathbf{x}), \sigma)$. Similarly, for logistic regression, we assume a label l is distributed, conditioned on input $\mathbf{x}$ with parameters $\mathbf{w}$, according to $(l \mid x; \mathbf{w}, \mathbf{f}) \sim \text{Bernoulli}(\mathbf{w}^\top \mathbf{f}(\mathbf{x}))$. Given these assumptions, we find the maximum-likelihood estimate (MLE) of $\mathbf{w}$ for both models using stochastic gradient descent (SGD).

We introduce a feature grammar into the linear models by applying its rules to highly weighted features over successive iterations of SGD, adding the resulting features to the vocabulary, and continuing to descend the gradient on the expanded vocabulary. More precisely, a feature grammar consists of a set of production rules $R = \{r : 2^{(X \to \mathbb{R})} \to 2^{(X \to \mathbb{R})}\}$, where a rule $r \in R$ takes a set of features $F = \{f : X \to \mathbb{R}\}$, and produces a set of more complex features $r(F)$. For example, a grammar rule might take two binary logical features and produce their conjunction or disjunction. At each iteration of SGD, given a threshold hyper-parameter t, we apply R to the set of highly weighted features $F_{t,\mathbf{w}} = \{f :| w_f |> t\}$ from the current vocabulary, and we add the resulting set $R(F_{t,\mathbf{w}})$ of features to the model.[2] We also add a feature-selecting $l1$ regularizer [5] to the loss, which allows the training procedure to prune out irrelevant features from the expanded set as the optimization procedure searches the feature space.

For this project, we have re-implemented these models in PyTorch using AdaGrad in place of vanilla SGD [12]. This required implementing an l_1-regularization option for the existing PyTorch AdaGrad implementation.

2.2 Tree-structured Architectures through Model Grammar

Given our use of the PyTorch neural network library, we can extend our features grammars to model grammars which produce network architectures. As a first attempt in this direction, we implement a grammar that explores the space of architectures that compute linear combinations of deep binary tree-structures, as shown in Figure 1. Initially, we start with a linear architecture (linear or logistic regression) with input features $\mathbf{f}_0$. This initializes the model's forest to a collection of depth-zero trees consisting of the input vocabulary— $F := \mathbf{f}_0$. Then, we apply the algorithm described in Section 2.1 to the the model with forest vocabulary F, except our rule set R consists of a rule which constructs new tree-structured features to add to F, and these new features contain their own new model parameters . In particular, the grammar R contains a single rule r which computes $r(F_{t,\mathbf{w}}) = \{f(\mathbf{x}) = \tanh(0, w_{(i,j),0} + w_{(i,j),1} f_i(\mathbf{x}) + w_{(i,j),2} f_j(\mathbf{x})) \mid f_i, f_j \in F_{t,\mathbf{w}}\}$ where each $w_{(i,j),k}$ is a new parameter added to the model. More informally, our grammar produces features which are $\tanh$ non-linearities applied to linear combinations of pairs of existing features. Recursively, applied within the algorithm from Section 2.1, this grammar results in a network architecture that is a linear combination of a forest of binary tree-structured features—where the trees contain their own model parameters for the linear combinations performed at each level. We chose this grammar for its construction of interpretable, shallow tree-structures over the seed feature set.[3]

[2] We realize that changing the feature vocabulary during SGD renders the optimization problem ill-defined. Nevertheless, there might still be a way to reformulate the problem such that a well-defined objective can be recovered. Furthermore, even if we cannot characterize the objective function, the learning algorithm might still be useful.

[3] In practice, this is actually implemented by a series of masked densely connected layers. We use masks to impose the binary tree structure over the features, and we also use masks to control which tree structures are added to the architecture by the grammar. As the grammar adds new tree features, more of parameters in the dense layers are unmasked.

3 Related Work

Our feature grammar models share motivations with work on traditional feature selection [9]. Namely, we desire automated methods for learning small, easily-interpretable, but highly accurate models on prediction tasks where there is a large space of potentially relevant input features. We borrow ideas from traditional feature selection work by introducing new features produced by the grammars analogously to forward selection (as described in [9]), except that we do not retrain until convergence before adding each new feature. Instead, we perform gradient updates under traditional feature-selecting $l1$-regularized mean-squared loss [5], which allows our algorithm to prune out irrelevant features introduced by the grammar without running until parameter convergence.

As a generalization of feature selection, structure learning for probabilistic graphical models also seems related to our methods [11, 10]. Traditional feature selection models can be thought of as solving the sub-task of structure learning where the goal is just to construct the Markov blanket for a single variable of a network [4]. This suggests that our grammar methods might be thought of as finding a Markov blanket for some variable when the graphical model contains a potentially infinite number of other variables that can be constructed from a grammar.

Our feature grammar models also resemble models from other learning paradigms like decision trees [1] and inductive logic programming (ILP) [2]. These paradigms impose some logical structure over the input features to approximate a function which computes the output class or label, and such logical structure is equivalent to logical features possibly constructed by a feature grammar. Most notably, our use of a feature grammar within linear models is similar to "structural logistic regression" [7], where new features are constructed through ILP, and selected by successive logistic regression models through forward selection. Our methods differ from this in that we do not retrain successive models until convergence, and we do not limit our models to the first-order logic formalism for features.

Our generalization from "feature grammar" to "model grammar" borrows ideas about feature construction from work on deep neural networks [14]. The hidden layers of a neural network can be thought of as constructing useful complex features for a task. The models produced by our model grammars behave similarly, except that the grammars are conservative in their introduction of complex features, allowing the resulting architectures to maintain some interpretability while possibly avoiding some poor local optima. In particular, we use a grammar to produce linear combinations of shallow binary-tree architectures (as described Section 2.2). These architectures resemble the ensembles of decision trees learned by random forests [3].

4 Experiments

For our experiments, we use PyTorch to reimplement our linear feature grammar models frpm Section 2.1, and reproduce our results from [15] on a synthetic logic data set constructed from models containing simple property features. Next, we implement the tree structure grammar model from Section 2.2, and characterize its behavior on two new synthetic logic data set constructed with ground-truth models containing conjunctive features.

4.1 Linear Feature Grammar Models on data set R

As a first sanity-check test of our models and our l_1-regularized AdaGrad PyTorch implementation, we reproduce our results from [15] with the linear feature grammar models described in Section 2.1 on synthetic first-order logic data. Specifically, we construct a random synthetic data set $\mathbf{R}$ of 3000 examples labeled by a ground-truth linear model, and we partition this into an 80/20 train/dev split.[4] Each data example consists of a collection of three objects o_0, o_1, o_2, and each object is randomly assigned a subset of binary properties from $\{P_0, P_2, \ldots, P_4\}$. An object o in a synthetic example is given property P_i with probability 0.5. The ground-truth linear model randomly assigns a label to example $\mathbf{x}$ according to $l \mid \mathbf{x} \sim \mathcal{N}(\sum_{i=0}^{4} w_i P_i(o_0)_{\mathbf{x}}, 0.1)$ where $w_i = \frac{2^i}{10}$. This renders the properties of objects o_1 and o_2 irrelevant, and so our models must learn to only select features based on properties of o_0. Our traditional $l1$-regularized linear regression model is given a feature vocabulary consisting of all properties P_j on all three objects o_j. The "feature grammar" version of linear regression (i.e. "linemar gramression") described in Section 2.1 is given a seed vocabulary containing only $\{P_0(0), P_0(1), P_0(2)\}$, and a grammar rule $P_i(x) \rightarrow P_{i+1}(x)$, so it must discover the relevance of other properties using the grammar rule.

The top plot in the $\mathbf{R}$ column of Figure 2 shows the dev set loss over successive AdaGrad iterations for our PyTorch implementations of linear regression and its corresponding "feature grammar" variant (from Section 2.1). This shows the result of training AdaGrad with a batch size of 50, l_1 hyper-parameter 0.25, learning rate 1.0, and feature grammar

[4] We refer to the evaluation as "dev" here since we primarily use it as a sanity check for our model development, and it functions analogously to the dev set we used when testing the tree-grammar models described below.

threshold $t = 0.0$. We chose the l_1 hyper-parameter to yield a linear regression model that selects features with similar weights to the ground-truth model, as shown in the **R** column of Table 1. Note that our goal in this experiment is to perform a qualitative sanity check on the behavior of our new implementation of the linear grammar model, and so tuning this hyper-parameter while peeking at ground-truth is a form of cheating that is useful rather than counterproductive. The bottom plot in the **R** column of Figure 2 suggest that the feature grammar model is able to select the correct ground-truth features at a similar rate to the traditional feature selecting l_1 linear regression. [5]

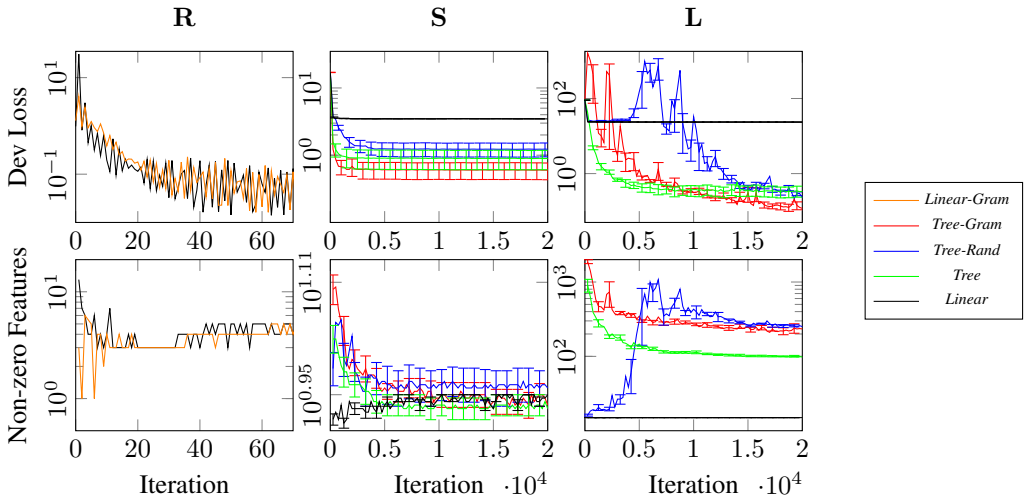

Figure 2: (**R** column) The dev losses and non-zero feature counts over AdaGrad iterations for our PyTorch reproductions of linear regression and its "feature grammar variant" (*Linear Gram*). (**S** and **L** columns) The dev losses and non-zero feature counts over AdaGrad iterations for a linear regression model (*Linear*), a full tree architecture (*Tree*), a randomly constructed tree (*Tree-Rand*), and a grammatically constructed tree (*Tree-Gram*) on the **S** and **L** conjunctive feature data sets. All vertical axes are in log scale.

4.2 Tree-structured Architecture Grammar Models

After reproducing our linear feature grammar models, we also implement the tree-structure architecture grammar method described in Section 2.2, and we gather evidence for several hypotheses about its behavior relative to several baseline models on two synthetic data sets **S** and **L**.

S (small) We construct synthetic data set **S** of 5000 examples labeled by a ground-truth model over a *small* number of conjunctive features, each containing two basic properties, and we partition this data into a 80/10/10 train/dev/test split. Each data example consists of a random subset of binary properties from $P = \{P_0, P_2, \ldots, P_{14}\}$, where each property P_i occurs in a given example with probability 0.5. From P, we construct a set of conjunctions $C = \{P_i \cdot P_j, P_i \cdot \neg P_j, \neg P_i \cdot P_j, \neg P_i \cdot \neg P_j \mid P_i, P_j \in P\}$, each involving two properties or their negations. The ground-truth linear model randomly assigns a label to an example $\mathbf{x}$ according to $l \mid \mathbf{x} \sim \mathcal{N}(\sum_{i=1}^{5} w_i C_i(\mathbf{x}))$ where $w_i = i$ and C_i is conjunction selected from C uniformly at random when the model is initialized. This will result in a data set where shallow tree-structured conjunctive features are highly predictive.

L (large) Similar to **S**, synthetic data set **L** contains 5000 labeled examples, partitioned into an 80/10/10 train/dev/test split. However, unlike **S**, the ground-truth model for **L** contains twice as many conjunctive features, and each feature contains up to four conjuncts. More precisely, each example in **L** consists of a random subset of binary properties from $P = \{P_0, P_2, \ldots, P_{14}\}$, where each property P_i occurs in a given example with probability 0.5. From P, we construct a set of conjunctions $C = \{\prod_{i=1}^{k} P_i^{1-n_i} \cdot (1 - P_i)^{n_i} \mid P_i \in P, n_i \in \{0, 1\}, k \in \{2, 3, 4\}\}$, each involving at most four properties or their negations. The ground-truth linear model randomly assigns a label to an example $\mathbf{x}$ according to $l \mid \mathbf{x} \sim \mathcal{N}(\sum_{i=1}^{10} w_i C_i(\mathbf{x}))$ where $w_i = i$ and C_i is conjunction selected from C uniformly at random when the model is initialized. This will result in a data set where deeper tree-structured conjunctive features are highly predictive.

[5] In a similar way, we also tested logistic regression and the feature grammar version of logistic regression against a ground-truth classification model. Unsurprisingly, the results for those models are similar to the results for the linear regression models.

R				**S**				**L**			
Truth		*Linear-Gram*		***Truth***		*Tree-Gram*		***Truth***		*Tree-Gram*	
P_4	1.60	P_4	1.55	$P_0 \wedge \neg P_{13}$	5.0	P_4	-2.97	$\neg P_{12} \wedge \neg P_5 \wedge \neg P_9$	10.0	$(P_5 \vee \neg P_{10}) \vee (P_9 \vee P_{13})$	-2.33
P_3	0.80	P_3	0.76	$P_{10} \wedge P_5$	4.0	$P_7 \vee P_{13}$	-2.50	$\neg P_5 \wedge P_{10} \wedge P_{13} \wedge \neg P_9$	9.0	$\neg(\neg P_1 \wedge P_{13}) \vee \neg(P_2 \vee \neg P_{13})$	-2.00
P_2	0.40	P_2	0.36	$P_6 \wedge \neg P_4$	3.0	$\neg P_0 \wedge \neg P_{13}$	-2.48	$P_2 \wedge \neg P_1 \wedge P_{13}$	8.0	$P_8 \wedge (\neg P_6 \wedge \neg P_9)$	1.85
P_1	0.20	P_1	0.14	$P_{13} \wedge P_7$	2.0	P_7	1.98	$\neg P_{11} \wedge P_{10}$	7.0	$(P_6 \vee P_7) \vee \neg(P_8 \vee \neg P_{10})$	1.82
P_0	0.10	P_0	0.05	$\neg P_{12} \wedge \neg P_0$	1.0	$\neg P_5 \vee \neg P_6$	-1.98	$P_2 \wedge P_{10} \wedge \neg P_3$	6.0	$(P_5 \vee \neg P_{10}) \vee \neg(P_{10} \wedge \neg P_{13})$	1.64

Table 1: ($\mathbf{R}$ column) Non-zero weighted property features $P_i(o_0)$ from the ground-truth model for the $\mathbf{R}$ data set described in Section 4.1, and their weights assigned by the linear feature grammar model *Linear-Gram*. ($\mathbf{S}$ and $\mathbf{L}$ columns) Highest magnitude weighted features in the ground-truth models, and logical form approximations to highest weighted features in learned *Tree-Gram* architectures on $\mathbf{S}$ and $\mathbf{L}$ data described in Section 4.2

On $\mathbf{S}$ and $\mathbf{L}$, we run the tree-structure grammar algorithm, *Tree-Gram* described in Section 2.2 with an initial input feature set consisting of basic properties in P. We compare to a *Linear* regression baseline over these same properties. We also compare to a random search baseline, *Tree-Rand*, which behaves like *Tree-Gram*, except that when the grammar rule fires, the architecture is extended with a random tree rather than one constructed by the grammar. Lastly, we compare to a full *Tree* baseline which is just the largest tree-architecture at a fixed depth trained from initialization rather than using the grammar. We train models for $20,000$ iterations using AdaGrad with batch size 50, learning rate 1.0. For *Tree-Gram* and *Tree-Rand*, we use a feature grammar threshold $t = 0.0$, and apply the architecture extension every 50 iterations for efficiency. We train each model over a grid of l_1 values in $\{0.001, 0.01, 0.1, 1.0\}$, and we average the losses for each model and hyper-parameter setting over 10 random weight initializations in the $\mathbf{S}$ condition and 5 random weight initializations in the $\mathbf{L}$ condition. We limit the maximum tree architecture feature depth for all methods to 1 for $\mathbf{S}$ and 2 for $\mathbf{L}$. In the right columns of Figure 2, we show the dev set losses and non-zero features for each model with the loss minimizing l_1 hyper-parameter values for $\mathbf{S}$ and $\mathbf{L}$.[6] Top weighted ground-truth features and approximate propositional logic representations of top tree-structures constructed by *Tree-Gram* are shown in Table 1.[7]

The results in Figure 2 and Table 1 have several interesting implications. First, the loss plot in Figure 2 for the $\mathbf{S}$ condition suggests that all tree-structure methods are able to outperform the *Linear* baseline by selecting a small set of tree-structure features. The plots for condition $\mathbf{S}$ also suggest that when the feature space is constrained to be small, *Tree-Rand* can perform just as well as *Tree-Gram* by randomly searching for features rather than using the grammar. However, the plots for condition $\mathbf{L}$ show that when the feature space is larger (due to the larger maximum tree depth), the grammar used by *Tree-Gram* helps to explore the feature space much more efficiently than *Tree-Rand*. Furthermore, most interestingly, the plots for $\mathbf{L}$ suggest that while *Tree-Gram* selects more features than the full *Tree* architecture, it also converges to a lower MSE loss than *Tree*. This suggests that slow addition of the tree-structures in *Tree-Gram* may avoid local optima encountered when training the full *Tree*. Finally, Table 1 shows that the approximate logical forms from the tree-structures discovered by *Tree-Gram* weakly match the ground-truth conjunctive features in $\mathbf{S}$, but this match is even weaker for the larger conjunctions used in $\mathbf{L}$. The weak match between the true and discovered features with the low loss of *Tree-Gram* suggest that the true feature set may be underdetermined by the data.

5 Conclusion and Future Work

In this project, we showed that our PyTorch reproductions of linear "feature grammar" models are able to recover models learned by traditional l_1-regularized linear regression. We also showed that a tree-structure model grammar can efficiently search a space of architectures to find a model with lower loss than the model initialized with all possible trees of a fixed maximum depth. Over the coming months, we will extend this work in several directions. First, we hope to test our methods on more synthetic data sets to improve the robustness of the results described above, and characterize the the conditions under which our grammar methods outperform training under an initially large full tree-architecture. Second, we hope to reproduce our feature grammar model results on the small organic molecules data set [16, 13] with our PyTorch implementation, and apply our new tree-architecture methods to this data as well. Finally, we hope to formally characterize the behavior of the model and feature grammar methods. If the grammar rules are applied stochastically, our method resembles a Markov chain Monte Carlo method [8], and so we might be able to characterize it as approximately sampling from some distribution over model parameters.

[6] We omit test set losses, but they take approximately the same values as dev set losses, which suggests that our l_1 grid search does not overfit the dev set.

[7] We construct approximate logic for the tree structures by computing the maximum and minimum possible input values to each unit, constructing truth-tables from these units under the assumption that an activation greater than 0.5 is "true", and deriving the propositional forms from these truth-tables.

Contributions and Acknowledgements

Bill was responsible for the algorithmic implementation. Stephan and Bill setup experiments to run, and collected and analyzed the results.

Tom Mitchell at CMU suggested the "feature grammar" idea to Bill when he was working for him a few years ago, and so he is responsible for inspiring the overarching theme of this work. We used the NIPS 2017 LaTeX template to build this document.

References

[1] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[2] Stephen Muggleton, Ramon Otero, and Alireza Tamaddoni-Nezhad. *Inductive logic programming*. Vol. 38. Springer, 1992.

[3] Tin Kam Ho. "Random decision forests". In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. Vol. 1. IEEE. 1995, pp. 278–282.

[4] Daphne Koller and Mehran Sahami. *Toward optimal feature selection*. Tech. rep. Stanford InfoLab, 1996.

[5] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288.

[6] Andrew Ng. "CS229 Lecture notes". In: *CS229 Lecture notes* 1.1 (2000), pp. 1–3.

[7] Alexandrin Popescul et al. "Towards structural logistic regression: Combining relational and statistical learning". In: (2002).

[8] Christophe Andrieu et al. "An introduction to MCMC for machine learning". In: *Machine learning* 50.1-2 (2003), pp. 5–43.

[9] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *Journal of machine learning research* 3.Mar (2003), pp. 1157–1182.

[10] Matthew Richardson and Pedro Domingos. "Markov logic networks". In: *Machine learning* 62.1-2 (2006), pp. 107–136.

[11] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[12] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[13] Raghunathan Ramakrishnan et al. "Quantum chemistry structures and properties of 134 kilo molecules". In: *Scientific data* 1 (2014).

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[15] Bill McDowell. *Feature-selecting Logistmar Gramression on Synthetic FOL Data*. 2017. URL: `https://github.com/forkunited/grampy/blob/master/src/main/resources/papers/feature-selection.pdf`.

[16] Bill McDowell and Stephan Eismann. *Molecular Substructure-selecting Limemar Gramression*. 2017. URL: `https://github.com/forkunited/grampy/blob/master/src/main/resources/papers/molecules.pdf`.

[17] Adam Paszke, Sam Gross, and Soumith Chintala. *PyTorch*. 2017. URL: `http://pytorch.org/`.