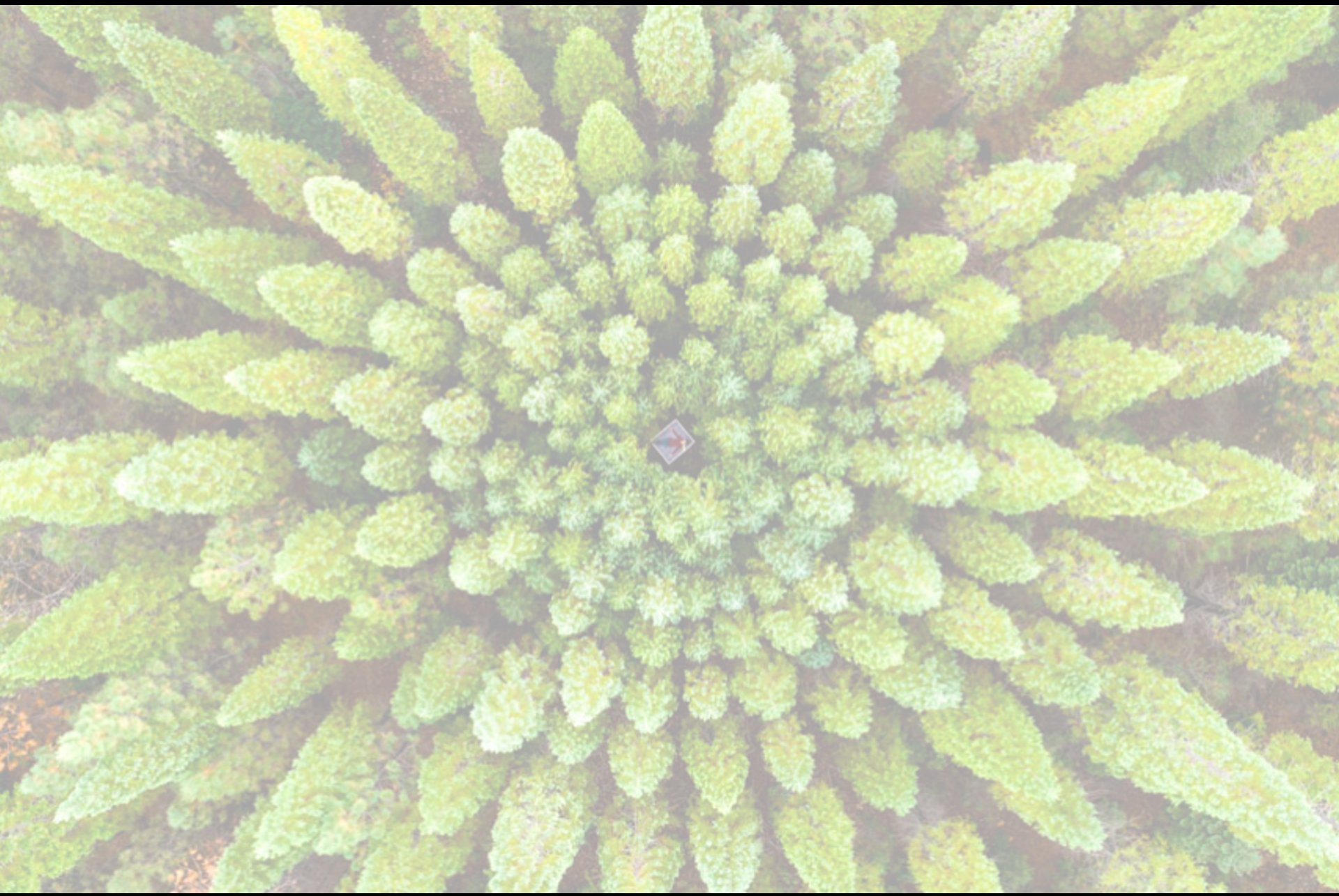
Lecture 16: Trees, Random Forests, and Boosting



Decision Trees – Instability

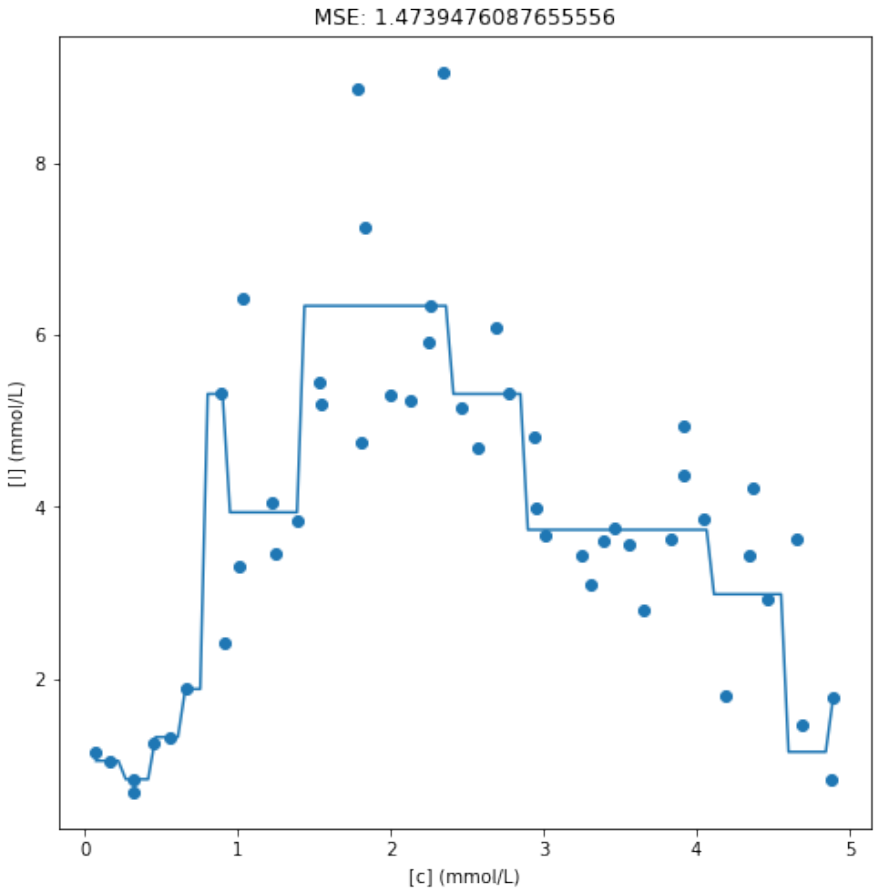
• Decision trees have an **instability** problem that we haven't covered yet.

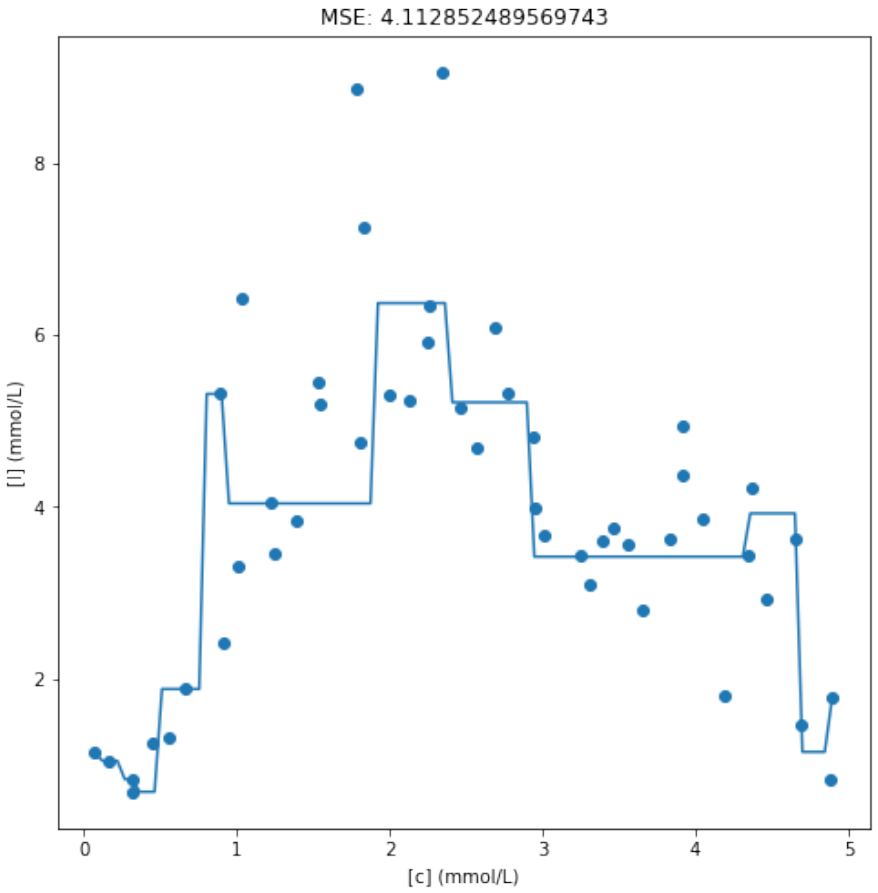
• By instability we mean that given small changes in training data, the topology, split values, and split features of a tree can change dramatically.

• This problem is most apparent in high dimensional problems, but even for our 1D regression problem, we can illustrate:

```
from sklearn.model_selection import train_test_split

# train_test_split is a helper function for generating random splits
x1_train, x1_test, y1_train, y1_test = train_test_split(x, y, test_size=0.2,\
                                                        random_state=135323)
x2_train, x2_test, y2_train, y2_test = train_test_split(x, y, test_size=0.2,\
                                                        random_state=235786)

# Train the first model
model = tree.DecisionTreeRegressor(max_depth=4)
model.fit(x1_train.reshape(-1, 1),y1_train)

# Train the second model
model = tree.DecisionTreeRegressor(max_depth=4)
model.fit(x2_train.reshape(-1, 1),y2_train)

…Plotting commands…
```

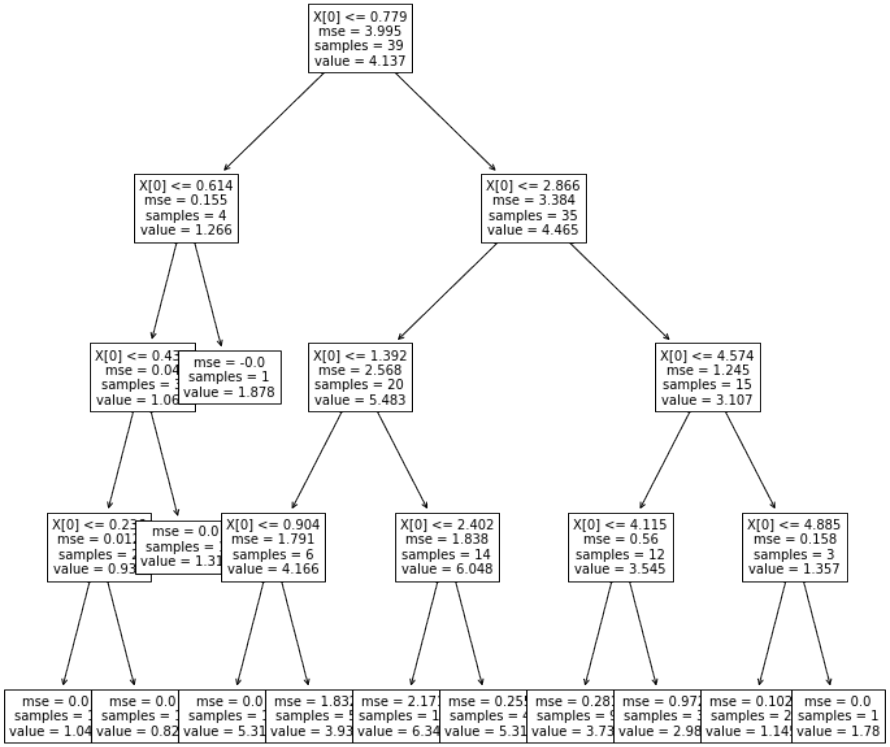
Decision Trees – Instability

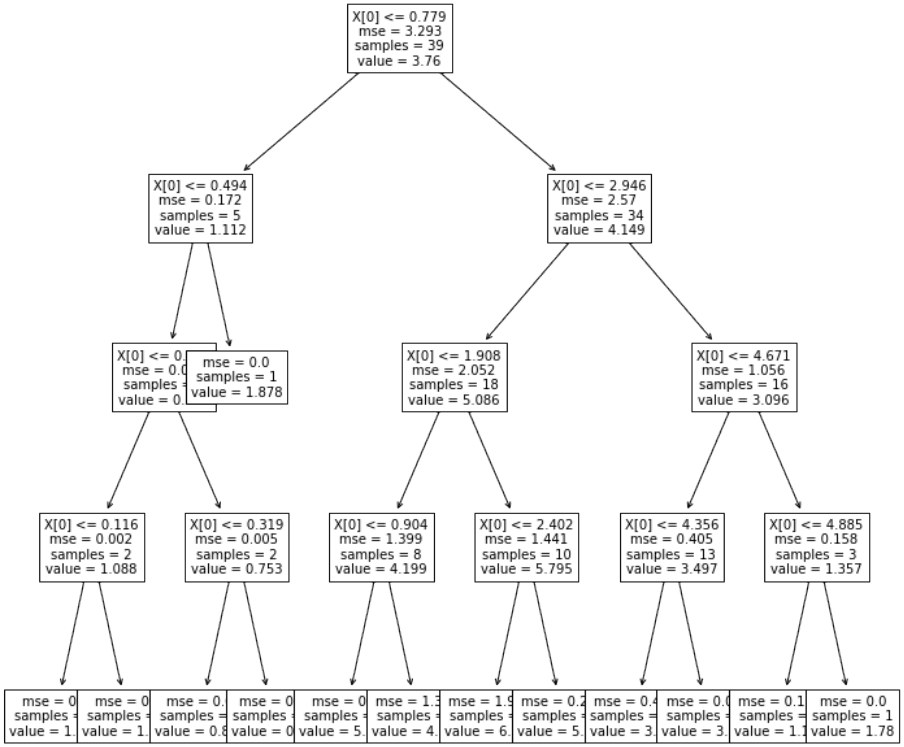




Similar shape, but there is a large difference in performance due to specific subset of training data.

Decision Trees – Instability





At the second and third levels we can see significant differences arising.

For higher dimensional problems, if the split variable at one of the levels change, then that propagates through the rest of the splits below it, which can lead to very different trees.

Decision Trees – Summary

Pros	Cons
• High interpretability. We have the tree structure and can inspect the important splits directly	• Unstable/High Variance (small changes in data can lead to dramatically different tree structures)
• Flexible data sources/multi-class problems. Mixed categorical and continuous features are fine.	• Prone to overfitting. Can be partially mitigated by pruning.
• Uniquely accomodates missing data.	• Lack of smoothness. Boxes aren't the best basis for all problems. Algorithms like MARS use an alternative basis.

The Wisdom of Crowds – Ensemble Methods

• The idea behind ensemble methods is that combining multiple relatively inaccurate models can perform better than a single relatively complex model:

$$f(x) = \sum_{i=1}^{\text{models}} \omega_i f_i(x)$$

where $f_i(x)$ is the prediction from an individual "weak learner" and ω_i is the relative weight given to each learner. In the simplest case, each model would get a vote and $\omega_i = 1/N_{models}$.

In general, if $f_i(x)$ are **independent** "*weak learners*", such a model will outperform any individual $f_i(x)$. This phenomenon is more broadly known as "the wisdom of crowds" and is the basis of "ensemble" prediction methods.

The Wisdom of Crowds – Ensemble Methods

• The idea behind ensemble methods is that combining multiple relatively inaccurate models can perform better than a single relatively complex model:

where $f_i(x)$ i… and ω_i is the relative wei… each model would get a …

In general, … model will outperform a… …lly known as ”the wisdom of crowds” and is the basis of “ensemble” prediction methods.

"Bagging" and Random Forests

• How do we get independent trees? Here the instability of the individual trees is a feature rather than a bug:

Bagging (Bootstrapping and Aggregation): Train individual trees on bootstrap resampled datasets (sampling with replacement). Aggregate predictions.

Random Feature Selection: Train individual trees on a randomly chosen subset of features.

• Model error can be estimated using the **out-of-bag error estimate** (i.e., average error across models on predictions for data point that weren't in their bag).

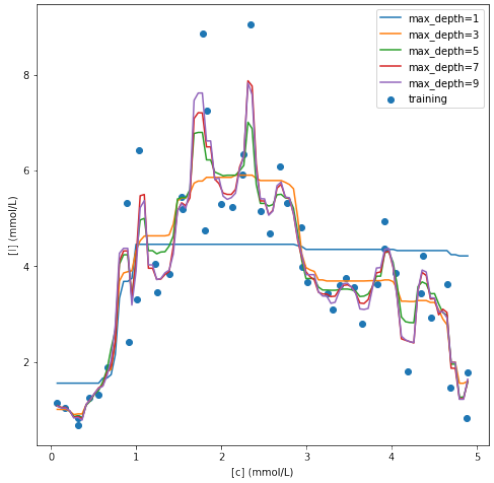
• 10-10,000 trees are commonly used in practice.

Bagged Trees - Example

"Bagged" trees only implements the bootstrap aggregation idea:

```
from sklearn.ensemble import BaggingRegressor

# Plot data
plt.figure(figsize=(8,8))
plt.scatter(x,y,label="training")
plt.xlabel("[c] (mmol/L)")
plt.ylabel("[I] (mmol/L)")

# Train decision tree regressors and fit
x_pred = np.linspace(min(x),max(x),100)
for i in range(1,10,2):
    model = BaggingRegressor(tree.DecisionTreeRegressor(max_depth=i),n_estimators=100)
    model.fit(x.reshape(-1, 1),y)
    plt.plot(x_pred,model.predict(x_pred.reshape(-1,1)),label="max_depth={}".format(i))
plt.legend()
plt.show()
```

Much smoother than individual trees because of the averaging effect

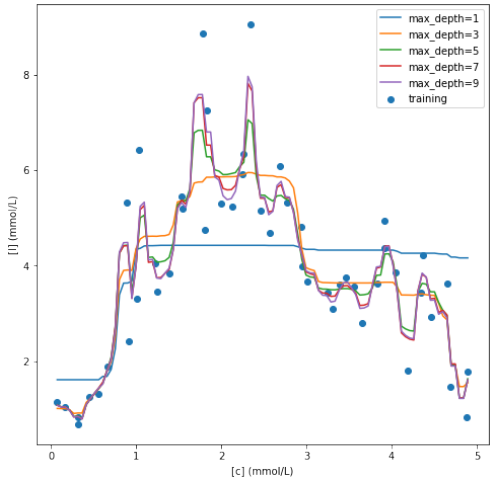


Random Forests - Example

Random forests combines bootstrap and random feature selection:

```
from sklearn.ensemble import RandomForestRegressor

# Plot data
plt.figure(figsize=(8,8))
plt.scatter(x,y,label="training")
plt.xlabel("[c] (mmol/L)")
plt.ylabel("[I] (mmol/L)")

# Train decision tree regressors and fit
x_pred = np.linspace(min(x),max(x),100)
for i in range(1,10,2):
    model = RandomForestRegressor(max_depth=i,n_estimators=1000)
    model.fit(x.reshape(-1, 1),y)
    plt.plot(x_pred,model.predict(x_pred.reshape(-1,1)),label="max_depth={}".format(i))
plt.legend()
plt.show()
```

Note: this case is 1D so the result is indistinguishable from bagged trees.



Random Forests – Additional Details

Pros	Cons
• **More robust against overfitting** • **Works better w/ complex data** • In practice, requires fewer hyperparameters (number of trees) • Adaptability (unsupervised & regression)	• Loss of interpretability • More expensive to train (but still less expensive than other models) • Predictions are likewise more expensive (not typically a bottleneck but it can be).

Random Forests – Additional Details

Pros	Cons
• More robust against overfitting	• Loss of interpretability

"Unfortunately the unstable models most helped by bagging are unstable because of the emphasis on interpretability, and this is lost in the bagging process."

Hastie et al. *Elements of Statistical Learning*

regression)

Boosting

Random forests are the prototypical example of ensemble methods. A similar process can be done using different types of classifiers and regressors

- When we train N_B models during bagging we are actually neglecting potentially useful information:
    - For each model we train, we can assess which datapoints are giving us trouble. In subsequent models we can increase the weight of those points in the loss function to promote getting them right.
    - There are several clever ways of updating these weights and how the models are combined to form a committee.
- AdaBoost was the first implementation of this idea, and Gradient Boosting is its generalization.

High Dimensional Example

Loading and cleaning the solvation free energy dataset:

```
import pandas as pd

# Load free energy of solvation data for small molecules in water
features = pd.read_csv("SAMPL_descriptors_df.csv") # Note will give a warning about some
of the features being mixed type.
labels = pd.read_csv("SAMPL_df_revised.csv")

# Only keep the features that can be converted to floats
keeps = []
for i in features.columns:
    try:
        features[i] = features[i].apply(float)
        keeps += [i]
    except:
        pass
features = features[keeps]
```

	iupac	smiles	expt	calc
0	4-methoxy-N,N-dimethyl-benzamide	CN(C)C(=O)c1ccc(cc1)OC	-11.01	-9.625
1	methanesulfonyl chloride	CS(=O)(=O)Cl	-4.87	-6.219
2	3-methylbut-1-ene	CC(C)C=C	1.83	2.452
3	2-ethylpyrazine	CCc1cnccn1	-5.45	-5.809
4	heptan-1-ol	CCCCCCCO	-4.21	-2.917
..	...	...	...	...
637	methyl octanoate	CCCCCCCC(=O)OC	-2.04	-3.035
638	pyrrolidine	C1CCNC1	-5.48	-4.278
639	4-hydroxybenzaldehyde	c1cc(ccc1C=O)O	-8.83	-10.050
640	1-chloroheptane	CCCCCCCCl	0.29	1.467
641	1,4-dioxane	C1COCCO1	-5.06	-4.269

High Dimensional Example

Training models:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import tree
from sklearn.metrics import mean_squared_error

np.random.seed(352809821) # for reproducibility

# Generate an 80:20 split for training and validation
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.2,random_state=135323)

# Initialize the models
# NOTE: You should experiment with hyperparameters in these initializations. Or implement
cross validation to search over hyperparameter values.
models = { "DT": tree.DecisionTreeRegressor() ,\
"RF": RandomForestRegressor(),\
"BR": BaggingRegressor(),\
"ADA":
AdaBoostRegressor(tree.DecisionTreeRegressor(max_depth=10),n_estimators=100,learning_rate
=0.1),\
"GBoost": GradientBoostingRegressor(max_depth=5) }

for i in models.keys():
    models[i].fit(X_train,y_train["expt"])
    train_err = mean_squared_error(y_train["expt"],models[i].predict(X_train))
    test_err = mean_squared_error(y_test["expt"],models[i].predict(X_test))
```

High Dimensional Example

Training models:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import tree
from sklearn.metrics import mean_squared_error

np.random.seed(352809821) # for reproducibility
```

See notebook for results and further discussion

```
# NOTE: You should experiment with hyperparameters in these initializations. Or implement
cross validation to search over hyperparameter values.
models = { "DT": tree.DecisionTreeRegressor() ,\
"RF": RandomForestRegressor(),\
"BR": BaggingRegressor(),\
"ADA":
AdaBoostRegressor(tree.DecisionTreeRegressor(max_depth=10),n_estimators=100,learning_rate
=0.1),\
"GBoost": GradientBoostingRegressor(max_depth=5) }

for i in models.keys():
    models[i].fit(X_train,y_train["expt"])
    train_err = mean_squared_error(y_train["expt"],models[i].predict(X_train))
    test_err = mean_squared_error(y_test["expt"],models[i].predict(X_test))
```