End-to-End Machine Learning: Doing Machine Learning

Glenn Bruns
CSUMB

Learning outcomes

After this lecture you should be able to write Numpy/Scikit-Learn code to:

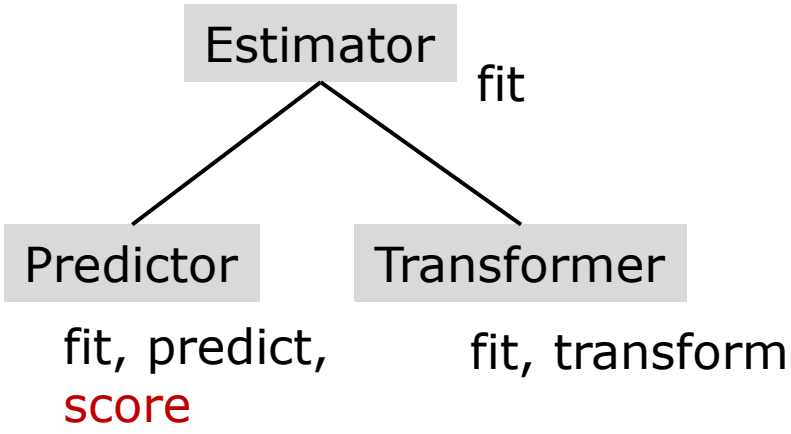
1. create test/training data sets
2. do machine learning with a Scikit-Learn predictor
3. perform cross validation
4. perform a grid search to fine tune a model

Building training and test sets

```
import pandas as pd
from sklearn.model_selection import train_test_split

dat = pd.read_csv(“cars.csv”)

# (any needed preprocessing goes here)

X = dat[['Price', 'Cylinder', 'Liter', 'Doors',
'Cruise', 'Sound', 'Leather']]
y = dat['Mileage']
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.3)
```

A good example of how parallel assignment is handy.

Doing machine learning with a predictor




```
from sklearn.linear_model import LogisticRegression

log_regr = LogisticRegression()
log_regr.fit(X_train, y_train)
y_pred = log_regr.predict(X_test)
lr_scores = log_regr.score(X_test, y_test)
```

Training, validation, and test sets

We've split data into training and test sets

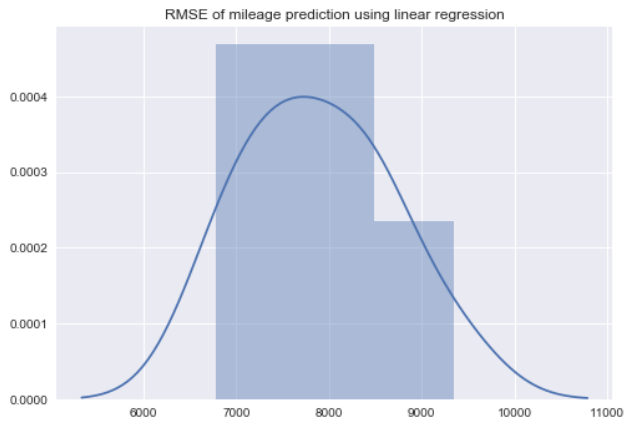
- A test set is used only to see how a model will generalize to unseen data
- Don't use test set to fine tune a model!

Question: can you name two ways to fine tune a model?

Cross-validation with Scikit-Learn

This is easy, because predictors have a standard interface.

```
from sklearn.model_selection import cross_val_score

log_regr = LogisticRegression()

scores = cross_val_score(log_regr, X_train, y_train,
                         scoring = “neg_mean_squared_error”, cv=10)

sns.distplot(np.sqrt(-scores))
```

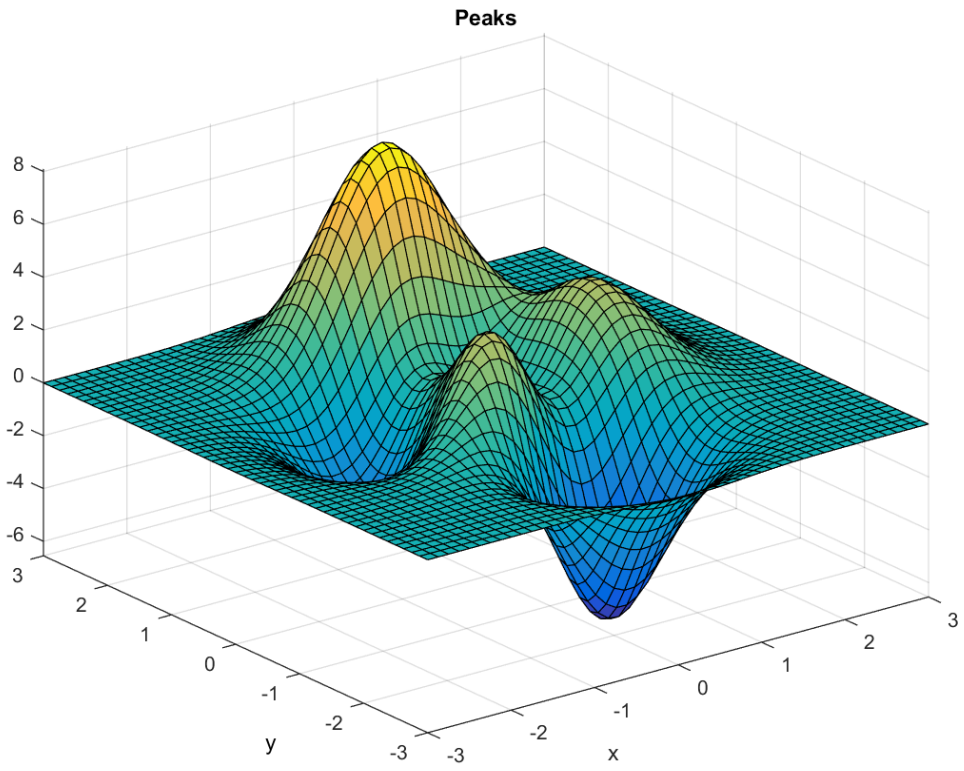


Question: how does the code change if we use tree regression?

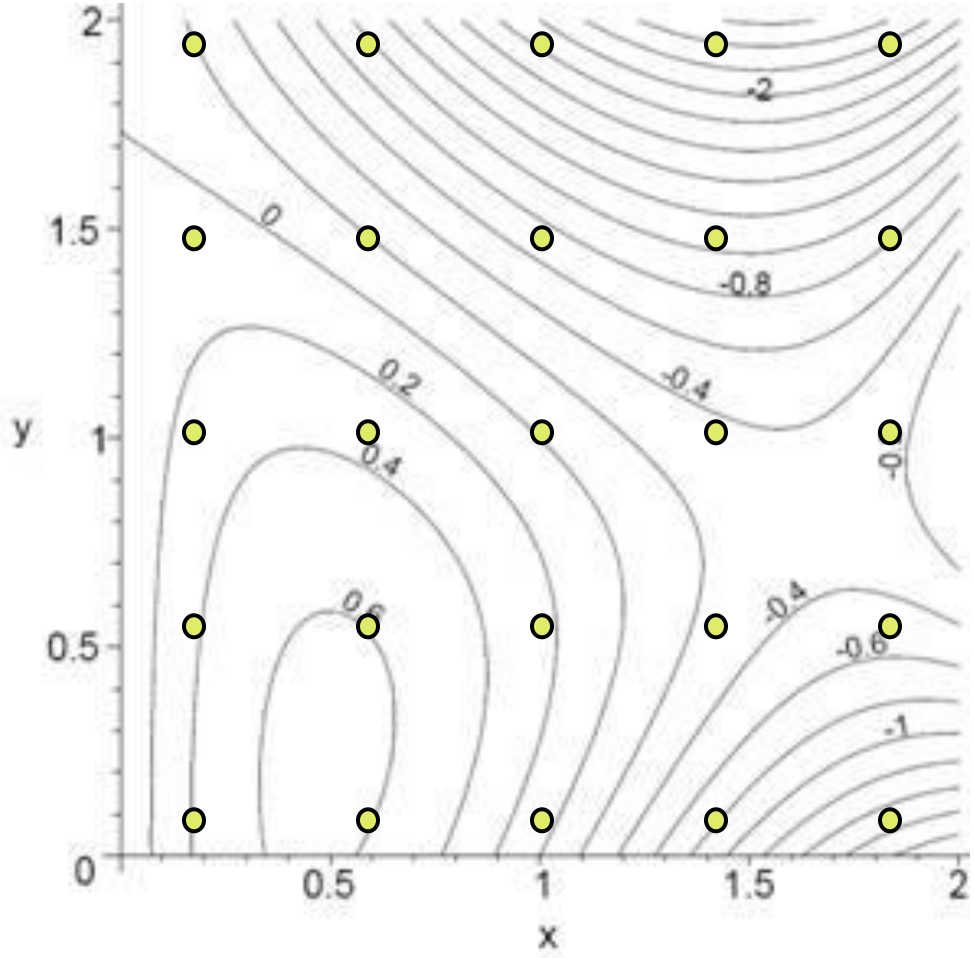
Finding best hyper-parameters

The RandomForest Regressor has 15 parameters that can be tweaked.

How to find the best combination of these hyper-parameters?



Grid search



If you test your machine learning algorithm at these points, will you find the global minimum?

Grid search with Scikit-Learn

```
param_grid = [
        {'n_estimators':[10,30,50], 'max_features':
             [3,5,7], 'max_depth': [5, 10, 20]},
        {'bootstrap': [False], 'n_estimators': [3,10],
             'max_features': [2,3,4]}]

# create the grid search object
regrCV = GridSearchCV(RandomForestRegressor(),
       param_grid, cv=5, scoring='neg_mean_squared_error')

# perform the grid search
regrCV.fit(X_train, y_train)

# use the best estimator
y_predict = regrCV.predict(X_test)  # handy!
```

grid search seeks value that maximize the scoring function

Seeing the best hyperparameters

```
regrCV.best_params_
Out[750]: {'max_depth': 10, 'max_features': 5, 'n_estimators': 50}

regrCV.best_estimator_
Out[751]:
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
           max_features=5, max_leaf_nodes=None,
           min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
           oob_score=False, random_state=None, verbose=0,
           warm_start=False)
```

Getting scores for each test

```
cv_res = grid_search.cv_results_
for mean_score, params in zip(cv_res["mean_test_score"],
                              cv_res["params"]):
    print(np.sqrt(-mean_score), params)
```

Running the code:

```
cv_res = grid_search.cv_results_
for mean_score, params in zip(cv_res["mean_test_score"],
cv_res["params"]):
    print(np.sqrt(-mean_score), params)
7517.91531228 {'max_features': 3, 'max_depth': 5, 'n_estimators': 10}
7511.71347022 {'max_features': 3, 'max_depth': 5, 'n_estimators': 30}
7406.62925797 {'max_features': 3, 'max_depth': 5, 'n_estimators': 50}
7452.83883348 {'max_features': 5, 'max_depth': 5, 'n_estimators': 10}
7337.76127881 {'max_features': 5, 'max_depth': 5, 'n_estimators': 30}
...
```

Summary

- We saw how to create training and test sets
- Scikit Learn's standardized API makes some things very easy
  - cross-validation
  - grid search for finding a good combination of hyper-parameter values for a ML algorithm