Lecture 16: Exam, Grid Search, K-Fold Cross Validation

CS 167: Machine Learning

Exam

Thursday, November 17th in class

25% of course grade

Closed-book

You may use

- one 8.5x11" sheet of paper with your own hand-written notes (can use both sides)
- a calculator (cannot use network or CAS features)

Exam

What's not on the exam: *Python*/*programming*

Question Types: (may or may not include)

- multiple choice
- fill-in-the-blank
- short answer
- pen-and-paper exercises like those we've done in class
- mark up some data plots

Some Material We've Covered: Algorithms

Algorithms

- Find-S
- Candidate Elimination
- k-Nearest-Neighbor, weighted k-Nearest-Neighbor
- Z-Score Normalization method
- ID3 Decision Tree (including entropy/info-gain calculations)
- Random Forests (Bagging, Ensemble learning)
- Perceptron training rule
- Gradient Descent
- Stochastic Gradient Descent
- Artificial Neural Network (sigmoid and multilayer Perceptron)
- Support Vector Machines
- Grid Search
- K-Fold Cross-Validation
- Principal Component Analysis

Some Material We've Covered: Terms to Know

- more specific/general than
- hypothesis (aka Model)
- consistent
- version space
- classification
- regression
- inductive bias
- target concept/function/column
- entropy
- information gain
- mean, standard deviation
- prior
- linear separator
- decision surface/boundary
- overfitting
- train and test set
- mean squared error
- margin
- kernel trick

Example Kinds of Questions

- Why normalize data for k-Nearest-Neighbor?
- Given some data in a table, compute best attribute based on information gain
- Given ANN diagram, show how examples would be classified
- What is the difference between Perceptron, Gradient Descent, and Stochastic Gradient Descent?
- Given data plot
  - draw linear separator with biggest margin
  - draw linear separator with trade-off between big margin and consistency
  - draw first principal component, second principal component
- How can we decide if we might be overfitting?
- If I think I'm overfitting with _____ algorithm, what should I do?
- Does smaller C lead to more consistency with training set or smoother decision boundary?
- How do I prepare this data for _____ algorithm?

Recall: How we've been finding good parameters

```
import pandas
from sklearn import linear_model as lm
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import grid_search

mpg_data = pandas.read_csv('auto-mpg.csv')

#everything except first and last column are predictors
mpg_preds = mpg_data.columns[1:-1]

#fill in missing data
mpg_data['horsepower'] = mpg_data['horsepower'].fillna(mpg_data['horsepower'].mean())

(mpg_train, mpg_test) = train_test_split(mpg_data,test_size = 0.2)

regr_alg = lm.SGDRegressor(eta0=.001, n_iter = 1000)

regr_alg.fit(mpg_train[mpg_preds],mpg_train['mpg'])

predictions = regr_alg.predict(mpg_test[mpg_preds])
print(metrics.mean_squared_error(predictions, mpg_test['mpg']))
```


Problem

We often have more than one parameter that needs to be tweaked for any given algorithm on a particular data set.

Grid Search: search through many combinations of parameters and use the best one

- simultaneously try different `eta0` and `n_iter`
- try different kernels, values of C, and γ with a support vector machine
- etc.

Grid Search with scikit-learn

```
parameters = {'eta0':[0.1, .001, 1e-6, 1e-9, 1e-12],
              'n_iter':[10, 1000, 100000]}

regr_alg = grid_search.GridSearchCV(lm.SGDRegressor(), parameters)

regr_alg.fit(mpg_train[mpg_preds],mpg_train['mpg'])

print(regr_alg.best_estimator_.eta0)
print(regr_alg.best_estimator_.n_iter)

predictions = regr_alg.predict(mpg_test[mpg_preds])
print(metrics.mean_squared_error(predictions, mpg_test['mpg']))
```


Notes on Grid Search

```
param_grid1 = {'eta0':[1e-6, 1e-9, 1e-12],
               'n_iter':[1000, 100000]}

param_grid2 = {'eta0':[0.1, .001],
               'n_iter':[10, 1000]}

parameters = [param_grid1,param_grid2]

regr_alg = grid_search.GridSearchCV(lm.SGDRegressor(), parameters)
```

- the `parameters` dictionary can be a list of dictionaries if you want to search multiple grids (say for different kernels which use different parameters
- uses a default score metric for each algorithm, but you can specify one
- Grid Search actually uses something called *K-Fold Cross Validation* instead of just a single test/train split

K-Fold Cross Validation

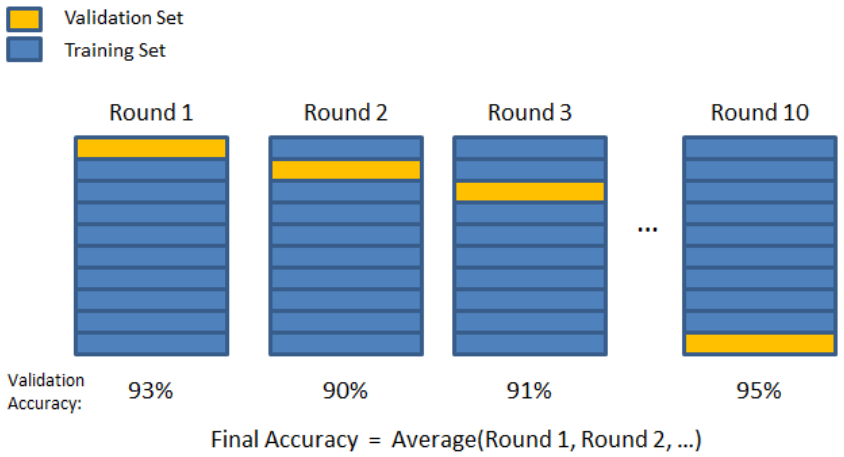

Image Credit: https://chrisjmccormick.wordpress.com/2013/07/31/k-fold-cross-validation-with-matlab-code/

Doing K-Fold Cross Validation in scikit-learn

```
from sklearn.model_selection import KFold
tot_mse = 0
folds = KFold(n_splits=4,shuffle=True)

#train/test give indices of folds
for train_idxs, test_idxs in folds.split(mpg_data):
    regr_alg = lm.SGDRegressor(eta0=1e-9, n_iter = 100000)
    train = mpg_data.iloc[train_idxs,:]
    test = mpg_data.iloc[test_idxs,:]
    regr_alg.fit(train[mpg_preds],train['mpg'])
    predictions = regr_alg.predict(test[mpg_preds])
    tot_mse += metrics.mean_squared_error(predictions,
                                          test['mpg'])

avg_mse = tot_mse/4
print(avg_mse)
```