COS 495 Precept 2 Machine Learning in Practice

Misha

Precept Objectives

- Review how to train and evaluate machine learning algorithms in practice.

- Make sure everyone knows the basic jargon.

- Develop basic tools that you will use when implementing and evaluating your final projects.

Terminology Review

Supervised Learning:

- Given a set of (example, label) pairs, learning how to predict the label of a given example.
- Examples: classification, regression.

Unsupervised Learning:

- Given a set of examples, learning useful properties of the distribution of these examples.
- Examples: word embeddings, text generation.

Other (e.g. Reinforcement, Online) Learning:

- Often involves an adaptive setting with a changing environment. Gaining some interest in NLP.

Example Problem: Document Classification

Given 50K (movie review, rating) pairs split into a training set (25K) and test set (25K), learn a function

$$f : \text{reviews} \mapsto \{\text{positive}, \text{negative}\}$$

For simplicity, represent each review as a Bag-of-Words (BoW) vector and each label as +1 or -1:

X_{train}: 25K V-dimensional vectors $x_1, \ldots, x_{25\text{K}}$.
Y_{train}: 25K numbers $y_1, \ldots, y_{25\text{K}} \in \{\pm 1\}$.

Approach: Linear SVM

- We will use a linear classifier:

$$f(x) = \text{sign}\left(w^T x\right), \quad w \in \mathbb{R}^V$$

- We will target a low hinge loss on the test set:

$$\sum_{(x,y)\in(X,Y)_{\text{test}}} \max\left\{0, 1 - y \cdot w^T x\right\}$$

Regularization

- If the vocabulary size is larger than the number of training samples then there is an infinite number of linear classifier that will perfectly separate the data. This makes the problem ill-posed.

- We want to pick one that generalizes well, so we use regularization to encourage a ‘less-complex’ classification function:

$$w^T w + C \sum_{i=1}^{25\mathrm{K}} \max\left\{0, 1 - y_i \cdot w^T x_i\right\}, \quad C \in \mathbb{R}_+$$

Regularization

```
>>> from sklearn.svm import LinearSVC
>>> svm=LinearSVC(C=1E-2).fit(Xtrain,Ytrain)
>>> svm.score(Xtrain,Ytrain)
0.8835260115606936
>>> svm.score(Xtest,Ytest)
0.7764964305326743
>>> svm=LinearSVC(C=1.0).fit(Xtrain,Ytrain)
>>> svm.score(Xtrain,Ytrain)
0.9986994219653179
>>> svm.score(Xtest,Ytest)
0.7819879187259473
>>> svm=LinearSVC(C=1E2).fit(Xtrain,Ytrain)
>>> svm.score(Xtrain,Ytrain)
0.9997109826589595
>>> svm.score(Xtest,Ytest)
0.7462932454695225
```

Cross-Validation

Validation:

- To determine C, we hold out some (say 5K examples) of our training data in order to use it as a temporary test set (also called ‘dev set’) to test different values of C.

Cross-Validation:

- Split data into k dev sets (‘folds’) and determine C by holding out each of them one a time and averaging the result.

Parameters are often picked from powers of 10 (e.g. pick the best-performing C out of 10^{-2}, … , 10^{2})

Evaluation Metrics: Accuracy

- Although we target a low convex loss, in the end we care about correct labeling alone. Thus for results we report the average accuracy:

$$\frac{1}{25\text{K}} \sum_{(x,y)\in(X_{\text{test}},Y_{\text{test}})} \mathbf{1}_{\{f(x)=y\}}$$

$$\text{where} \quad f(x) = \text{sign}\left(w^T x\right)$$

Evaluation Metrics: Precision/Recall/F_1

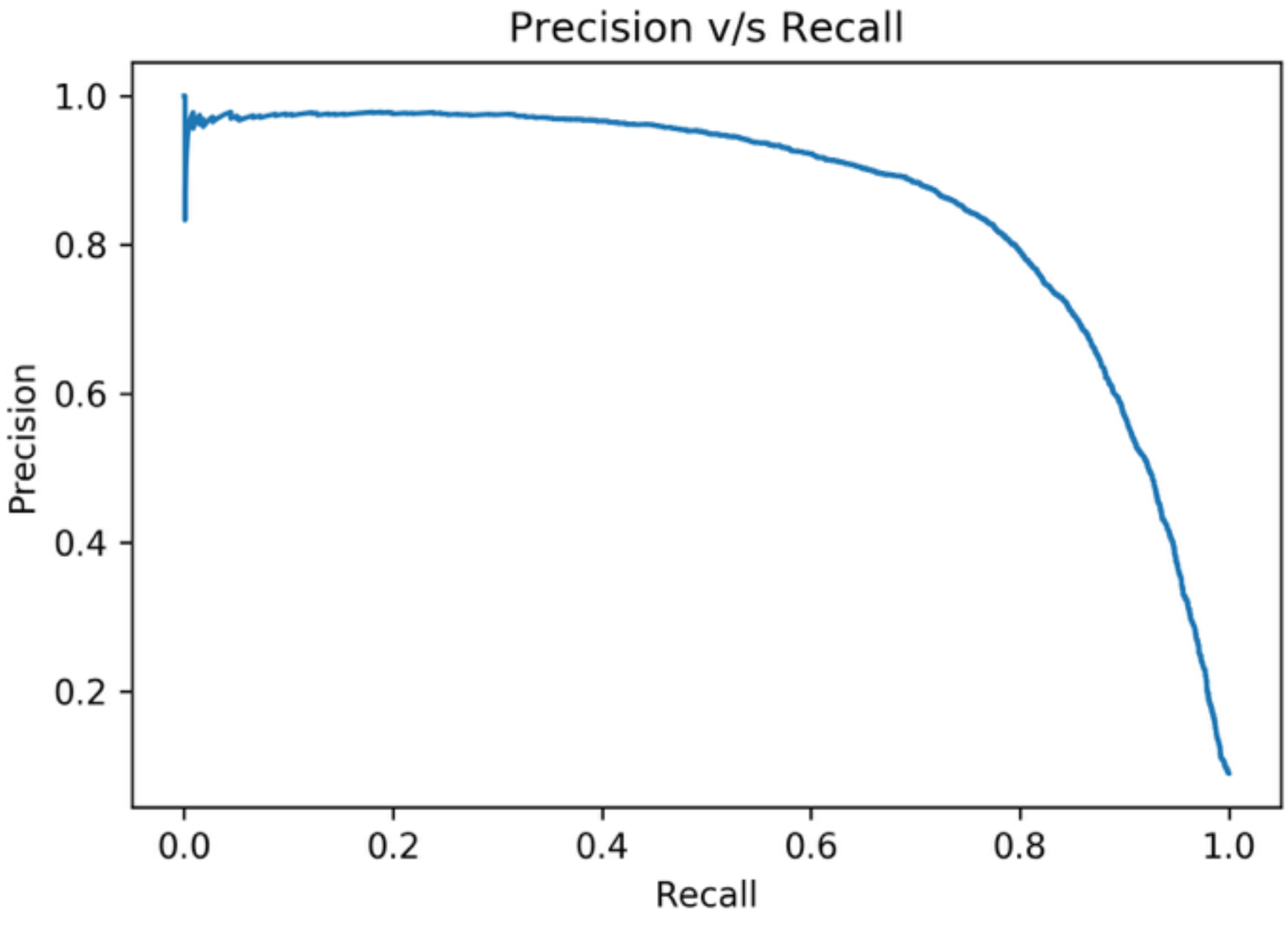
Sometimes, average accuracy is a poor measure of performance. For example, say we want to detect sarcastic comments, which do not occur very often, and learn a system that marks them as positive.

$$\text{precision} = \frac{\#\text{ True Positives}}{\#\text{ True Positives} + \#\text{ False Positives}}$$

$$\text{recall} = \frac{\#\text{ True Positives}}{\#\text{ True Positives} + \#\text{ False Negatives}}$$

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Precision v.s. Recall



Example Problem: Document Similarity

Given a set of (sentence-1, sentence-2, score) triples split into a training set (5K) and a test set (1K), learn a function:

$$f : \text{sentences} \times \text{sentences} \mapsto \mathbb{R}$$

Approach: Regression

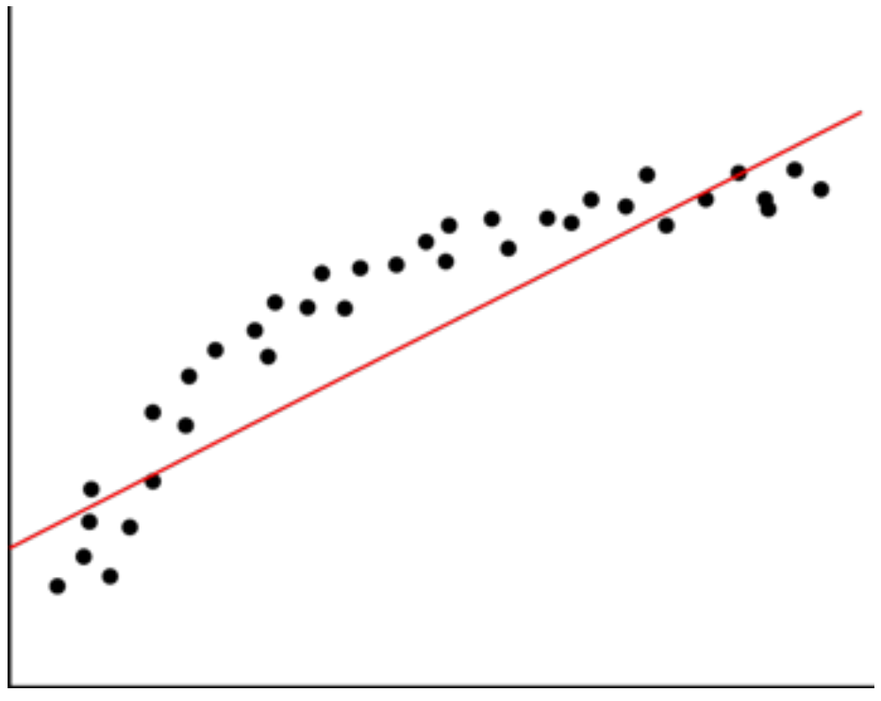
- Represent each pair of documents as a dense vector and minimizes the mean-squared-error between the function output and the score:

$$\frac{1}{10\text{K}} \sum_{i=1}^{10\text{K}} \|y_i - f(x_i)\|_2^2$$

- Tricky part is determining the function: linear, quadratic, neural network?

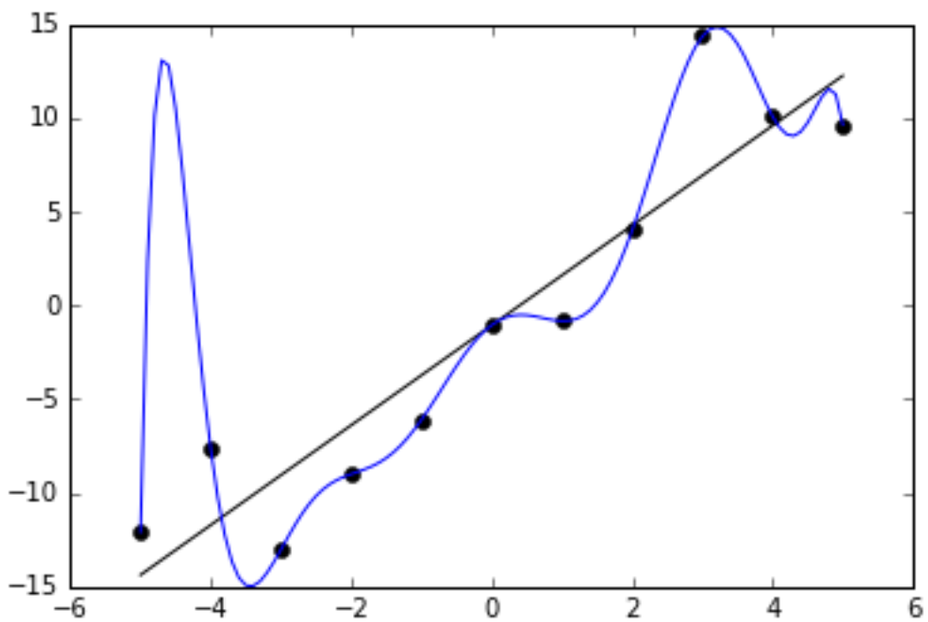
Under-fitting

- Under-fitting occurs when you cannot get sufficiently low error on your training set.
- Usually means the true function generating the data is more complex than your model.



Over-fitting

- Overfitting occurs when the gap between the training error and the test error (i.e. ‘generalization error’) is large.
- Can occur if you have too many learned parameters (as we saw in the BoW example).



Finding a Good Model

- Regularization: encourages simpler models and can incorporate prior information.

- Cross-validation: determine optimal model capacity by testing on held out data.

- Information criteria (Akaike, Bayesian)

What Changes When We Switch to Deep Learning?

More hyperparameters:

- Learning rate, number of layers, number of hidden units, type of nonlinearity, …
- Sometimes cross-validated, oftentimes not.

Higher model capacity:

- Deep nets can fit any function.
- Various regularization methods (dropout, early stopping, weight-tying, …)

Mini-batch Learning

Useful Tips in NLP: Sparse Matrices

- Often we deal with sparse features such as Bag-of-Words vectors. Storing dense arrays of size 25K x V is impractical.

- Sparse matrices (e.g. in scipy.sparse) allow usual matrix operations to be done efficiently without massive memory overhead.

Useful Tips in NLP: Feature Hashing/Sampling

- In some settings we have too many different features to handle (e.g. spam filtering, large corpus vocab).
- Can deal with this by min counting, but this discards data and is hard to use in an online setting.
- Different approaches:
  - Feature hashing: randomly map features to one of a fixed number of bins (used in spam filtering).
  - Sampling: only consider a small number of features when training (used for training word embeddings).