Data 100 Spring 2019 Final Exam Reference

Sampling

Simple Random Sample (SRS)

Prob. of 1 in sample of k from population of n:

$$P = \frac{k}{n} = \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + ... + \frac{1}{n-k+1}$$

Prob. of 1 from k samples selected from n:

$$P = \frac{\binom{n-1}{k-1}}{\binom{n}{k}}$$

Cluster Sample

Divide population into cluster, SRS to select cluster(s)

Prob. of 1 in sample = prob of 1's cluster in sample of k cluster from pop

$$P = \frac{k \text{ clusters}}{n \text{ clusters of population}}$$

Stratified Sample

Split into strata, SRS in **EACH** stratum

Prob of 1 in sample = prob of 1 selected from strata of n

$$P = \frac{1}{n}, \text{ where } n = \text{strata size}$$

Multi-Stage

Conditional event, SRS to select group, SRS to sample within selected groups

$$P(Stage2|Stage1)$$

Pandas

```
#Data Frames
df.loc[row, col] #row/column names
df.loc[[name1, name2]] -> dataframe
df.loc[name1] -> series
df.iloc[row, col] #row/column indices

#Series
s.value_count() # unique value counts

grouped = df.groupby(by)

#Aggregate groupby object
        grouped.agg(func)
        func: max, min, fisrt, last, head
                (lambda sf: ...)
                # manipulate subframe

#Filtering groupby object
        grouped.filter(func)
        func: (lambda sf: ...)
        # filtering condition

#merge
        df1.merge(df2, left_on=, right_on=)
```

SQL

Basic Syntax

```
SELECT  <columns>
FROM <relations(s)/tables>
[<join type> JOIN <relation/table>]
[ON <predicate>]
[WHERE <predicate>]
[GROUP BY <column(s)>] [HAVING <predicate>]
[ORDER BY <column(s)>] [LIMIT <number>]

ORDER BY RANDOM LIMIT # = random sample
```

Joins

Returns NULL if not found

INNER: Intersection

LEFT OUTER: keep rows on left

RIGHT OUTER: keep rows on right

WHOLE OUTER: everything

CASE Statement

```
CASE WHEN <predicate> THEN <rv_pred>
     ELSE <rv_else>
     END
```

RegEx

Characters

```
\d: Digits [0-9]
\w: Alphanumeric [a-zA-Z0-9]
\s: White space
\D,\W,\S: Inverses of respective groups
. : any character
\ : escape to match literals
```

Quantifiers

```
* : 0 or more times
+ : 1 or more times
? : exactly 0 or 1 times
{m} : exactly m times
{m, n} : between m to n times
{m, } : at least m times
{ , n} : at most n times
```

Anchors

```
^ : beginning of line
$ : end of line
```

Grouping

```
(...) : match all characters in capturing group
[^...] : excludes all specified
```

In Python

```
import re
re.match(pattern, string) # 1 match from beginning
re.search(pattern, string) # 1 match anywhere
re.findall(pattern, string) # list of all matches
re.sub(pattern, repl, string) # replace patterns
re.split(pattern, string) #split by pattern

s.str.extract(r'[group]') # extract by group
# will only return grouped regex match
```

Kernel Density Estimation

Smooth function to estimate probability distribution:

Center kernel at every point

add function together and take average

α : standard deviation of each kernel (large: spread apart, low/flat peak; small: centered, tall peak)

Gaussian

$$k_\alpha(x, z) = \frac{1}{\sqrt{2\pi\alpha^2}} e^{-\frac{(x-z)^2}{2\alpha^2}}$$

Decrease α decreases smoothness of plot.

Boxcar

$$\begin{cases} \frac{1}{\alpha} & -\frac{\alpha}{2} \le x - z \le \frac{\alpha}{2} \\ 0 & else \end{cases}$$

Probability distribution area = 1

rectangle: $\frac{1}{\alpha} \cdot \alpha$; bigger α more accurate

---

Dimensionality Reduction & PCA with SVD

Finding columns that are linearly independent in a design matrix (gives $Rank(X)$):

$$X = \mathbf{u}\Sigma\mathbf{v}^\top$$

Where,

$X : n \times d$, normalized original dataset ("feature matrix")

$\mathbf{u} : n \times r$, left singular vectors, orthonormal

$\Sigma : r \times r$, diagonal matrix of singular values. The k-th singular value is the sum of the squares of the projections of the points to the line determined by the k-th singular vector.

$\mathbf{v}^\top : r \times d$, right singular vectors, orthonormal

Directions

$\mathbf{v1}$ = the unit vector $\mathbf{v}$ that maximizes $|X\mathbf{v}|$

$\mathbf{v2}$ = the unit vector $\mathbf{v}$ that is orthogonal to $\mathbf{v1}$ and maximizes $|X\mathbf{v}|$

The i-th singular value of X, or $\sigma_i(X)$, is given by $|X\mathbf{v_i}|$. The first k columns of $X\mathbf{v}^\top$ (or $\mathbf{u}\Sigma$) are the first k principal components

Computations

1. Centering Matrix:

```
n = X.shape[0]
X_transformed = (X - np.mean(X, axis=0))/np.sqrt(n)
```

2. Use SVD to find PCs:

```
u, s, vt = np.linalg.svd(X_transformed, full_matrices=False)
```

3. Selecting PCs

```
(u * s)[:,0]        X @ vt[0,:]        (X @ vt.T)[:,0]
```

Visualizations

Things to Consider

Granularity: what each row represents

Faithfulness: does data accurately capture reality

Temporality: how does data situate in date/time

Scope: coverage of data in relations to analysis (completeness)

Plots

Single Discrete: dot plot/bar plot

Single Continuous: strip plot (small), density plot/bar plot (big)

Multiple Continuous: Scatter plot

Discrete vs. Continuous: Box plot, overlay density plot, violin plot

2 Discrete vs. 2 Continuous: Sub-scatter, color, symbols, overlays

Probability, Expectations & Variance

Sampling

Without replacement $= \binom{n}{r} = {}_nC_r = \frac{n!}{n!(n-r)!}$ ways

Prob of picking 1: $\frac{(n-1)_nC_r}{{}_nC_r}$

With replacement $= n^r$ ways

Prob of picking 1: $1 - \frac{(n-1)^r}{n^r}$

Expectation

$E[X] = \Sigma(p(x_i) \cdot x_i)$

$E[X+Y] = E[X] + E[Y]$

$E[cX] = c \cdot E[X]$

$E[XY] = E[X] \cdot E[Y]$ (indep.)

Median Absolute deviation: $Median[X - Median[X]]$

Variance

$Var[X] = E[(X - E[X]^2] = E[X^2] - E[X]^2$

$Var[aX] = a^2 \cdot Var[X]$

$Var[X+Y] = Var[X] + Var[Y] + 2Cov[X,Y]$

$Cov[X,Y] = Var[X] \cdot Var[Y] = E[XY] - E[X] \cdot E[Y]$

Same mean/variance $\neq$ same data

Bernoulli

$E[X_i] = p$ and $Var[X_i] = p(1-p)$

Calculating variance: $Var[Y] = Var(\frac{1}{n}\sum x_i)$

or, for Bernoulli: $Var[X] = np(1-p), y = \frac{1}{100}(x)$

Probability Distribution

Gaussian

$$f_N(x : \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Bernoulli

$$f_{\text{binary}}(y : n, p) = {}_nC_y \cdot p^y(1-p)^{n-y}$$

Linear Algebra Properties

$\mathbf{a} \cdot \mathbf{a} = \mathbf{a}^\top\mathbf{a}$

$\mathbf{a} \cdot \mathbf{b} = \mathbf{b}^\top\mathbf{a} = \mathbf{a}^\top\mathbf{b}$

$(AB)^\top = B^\top A^\top$

$\nabla_x \mathbf{a}^\top\mathbf{x} = \nabla_x \mathbf{x}^\top\mathbf{a} = \mathbf{a}$

$\nabla_x \mathbf{x}^\top A\mathbf{x} = (A + A^\top)\mathbf{x}, = 2A\mathbf{x}$ if $A = A^\top$ symmetric

Projection Matrix: $A^2 = A$

Loss & Risk Optimization

Loss Functions

Measuring loss at a specific point/observation:

$L_1 = |y - \hat{y}|$, minimized by median

$L_2 = (y - \hat{y})^2$, minimized by mean

Huber: $\begin{cases} \frac{1}{2}(y-\hat{y})^2 & |y-\hat{y}| \le \alpha \\ \alpha|y-\hat{y}| - \frac{1}{2}\alpha & else \end{cases}$

To find empirical risk, average loss from every point!

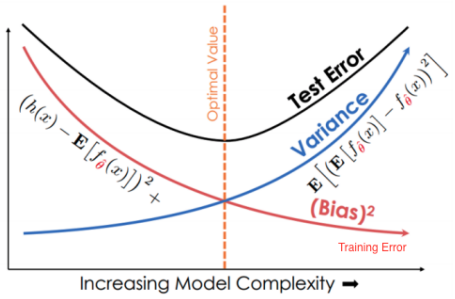
Minimizing/estimating parameter: Find Derivative of Risk (β **is Constant!**) Set Derivative to 0 Solve forβ

Regularization, Bias & Variance Trade-off

Risk is the average loss over an entire set of observations:

Risk $= E[(\mathbf{y} - f(\mathbf{x}))^2] = (E[f(\mathbf{x})] - \mathbf{y})^2 + E[(f(\mathbf{x}) - E[f(\mathbf{x})])^2]$

$= Bias^2[\mathbf{y}, f(\mathbf{x})] + Var[f(\mathbf{x})]$



Polynomial: higher degree, $Bias \downarrow, Variance \uparrow$, model complexity $\uparrow$

Regularization for Mean Squared Error Function

Regularize when **high loss, high variances**, controlling low bias

$\lambda \to 0, \hat{\theta} \to L_2$, little regularization

$\lambda \to \infty, \beta_i \to 0$, constant model, under-fit

$\lambda \uparrow \to Bias \uparrow \to Variance \downarrow$

Ridge (L_2): $\frac{1}{n}\|y - X\beta\|_2^2 + \lambda\sum\beta^2$

$\beta^* = (\mathbf{x}^\top\mathbf{x} - \lambda I)^{-1}\mathbf{x}^\top\mathbf{y}$

Distribute weight across related feature

Does not encourage sparsity **(does not set anything to 0)**

LASSO (L_1): $\frac{1}{n}\|y - X\beta\|_2^2 + \lambda|\beta|$

No Analytical Solution β^*

Encourages sparsity **setting some weight to 0, decrease some** (feature selection)

Linearity of features: x terms can be polynomial, but β must be in separate terms and unique to each x.

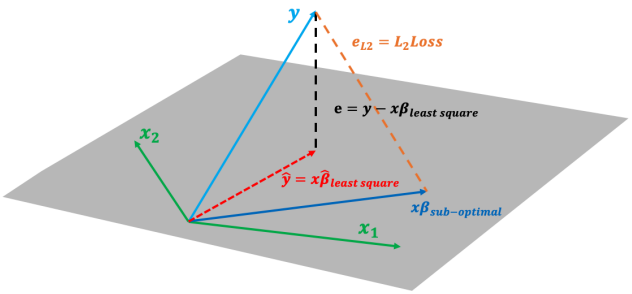
Convexity: All lines connecting two points of the graph appear above the graph.

Linear Regression & Least Squares

Linear Regression Model: $E[Y|X] = X^\top \beta$

Normal Equation:
$L(\beta) = \mathbf{y}^\top \mathbf{y} - 2\beta^\top \mathbf{x}^\top \mathbf{y} + \beta^\top \mathbf{x}^\top \mathbf{x}\beta$
$\nabla_\beta L(\beta) = \mathbf{x}^\top \mathbf{x}\beta - \mathbf{x}^\top \mathbf{y} = 0$
$\beta^* = (\mathbf{x}^\top \mathbf{x})^{-1}\mathbf{x}^\top \mathbf{y} = \sum_{i=1}^n \frac{x_i y_i}{x_i^2}$

Conditional Covariance:
$Cov[\hat{\beta}|\mathbf{x}] = \sigma^2(\mathbf{x}^\top \mathbf{x})^{-1}$, where $Var[Y|X] = \sigma^2$



*error/residuals need to be orthogonal to $Col(X)$ (aka MSE)
$X : n \times p$, to find some $\hat{\beta}$ that minimizes $\|e\|_2^2$
$\hat{\mathbf{y}}$: observed $\mathbf{y}$ projected on $Col(X)$, linear, orthogonal to residuals
Sum of residuals = 0

Gradient Descent

$$\beta^{t-1} = \beta^t - \alpha \nabla_\beta L(\beta^t)$$

α = learning rate
standard/batch: use all points
stochastic: pick random point to approximate batch
mini-batch: pick k random points

Algorithm:

```
def grad_desc(x, y, theta, iter=#, alpha=#):
    for i in range(iter):
        grad = dt(x, t, theta)
        theta = theta - alpha * grad
        loss = mse(model(x, theta[0], theta[1], y))
        theta_history.append(theta)
        loss_history.append(loss)
    return theta, theta_history, loss_history
```

Cross Validation

Fold = # of times splitting

Algorithm:

```
def CV_error(model, X_train, Y_train):
    kf = KFold(n_splits = 4)
    validation_errors = []

    for train_idx, valid_idx in kf.split(X_train):
        # Split Data
        split_X_train = X_train.iloc[train_idx]
        split_X_valid = X_valid.iloc[valid_idx]
        split_Y_train = Y_train.iloc[train_idx]
        split_Y_valid = Y_valid.iloc[valid_idx]

        # Fit Model
        model.fit(split_X_train, split_Y_train)

        # Compute Error
        Y_pred = model.predict(split_X_valid)
        error = rmse(Y_pred, split_Y_valid)

        valifation_errors.append(error)

    return np.mean(validation_errors)
```

Logistic Regression/Classification

Equations:

Logistic Regression Model: $E[Y = 1|X] = \sigma(X^\top \beta)$
$log(\frac{P(Y=1|X)}{P(Y=0|X)}) = X^\top \beta$
The intercept term helps us shift the center/inflection point of the sigmoid away from the origin.

Sigmoid Function: $\sigma(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}$
$\sigma(-t) = 1 - \sigma(t)$
$\sigma'(t) = \sigma(t)(1 - \sigma(t))$
$X^\top \beta \to \infty, \sigma(t) \to 1$
$X^\top \beta \to -\infty, \sigma(t) \to 0$

Cross Entropy Loss

$-\mathbf{y_i} log(\theta) - (1 - \mathbf{y_i}) log(1 - \theta)$

TFAE when $\theta = \sigma(\mathbf{x_i}^\top \beta)$:
$L(y, \theta) =$
$-\mathbf{y_i} log(\sigma(\mathbf{x_i}^\top \beta)) - (1 - \mathbf{y_i}) log(1 - \sigma(\mathbf{x_i}^\top \beta))$
$-\mathbf{y_i}\mathbf{x_i}^\top \beta + log(\sigma(-\mathbf{x_i}^\top \beta))$

Classifier Evaluation

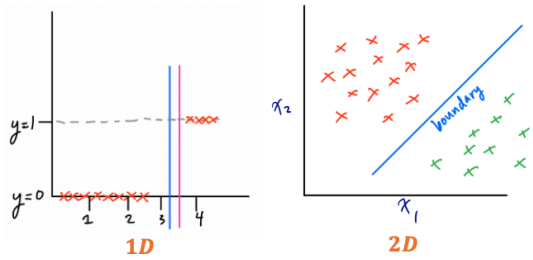
	1	0
1	TP	FP
0	FN	TN

accuracy: (TP+TN)/n
error: (FP+FN)/n
precision: TP/(TP+FP) (# of correct label over all true labels)
recall: TP/(TP/FN) (# of true labels over all actually true items)
Code:

```
np.count_nonzero(
    (y==y_pred) & (y_pred==1)  #TP
    (y==y_pred) & (y_pred==0)  #TN
    (y!=y_pred) & (y_pred==1)  #FP
    (y!=y_pred) & (y_pred==0)  #FN
)
```

Linear Separability

d-dimensional points, draw a $d-1$ dimension line/plane to separate points:



To be linearly separable, the data just needs to be separable on **one of the feature columns**.

* if data is linearly separable, then we can choose some β such that the cross entropy loss $\to 0$, and our optimal $\beta \to \infty$ (vertical line) (OR: our β's in the model will be very large, and will not be set to 0 by LASSO.) **Should regularize.**

Tree-Based Methods

Decision Trees

Each node represents a variable, and **splits based on variable's values**. Leaf node associated with **prediction values**
Training: Identify how tree should be structured
Prediction: start from root, leaf node signifies prediction

1. Selecting splits;
2. Decide whether to declare node to be terminal or continue;
3. Assign fitted values at each terminal node

Deciding Splits

Split at each node to maximize decrease in risk of current node (locally optimal)

Greedy Optimization: minimum risk occurs by taking a sub-optimal split at current node; depth first search computationally expensive

Classification: Impurity Measures as Risk Function (out of scope)

Regression: Squared Loss, Absolute Loss, Huber Loss

Stopping Splits + Prune to Avoid Overfitting

To prevent overfitting and classify over every single point:

Pruning: remove section of tree that has little effect

Set **maximum tree depth**, **minimum number of samples** required to split, or **maximum impurity threshold**

Ensamble Methods

Building multiple trees to avoid overfitting

Regression: average/median across tress (reduce variability + smooth regression surface)

Classification: majority vote

Bagging (Bootstrap Aggregating): same tree trained on multiple bootstrap sample (good when single tree overfit)

Boosting: Assign higher weights to misclassified points (combined weak learners to create 1 strong learner)

Difference in Prob: Bagging - each point has equal chance of resample; Boosting - weights determined by performance

Pros & Cons

Pros:

Highly interpretable

Performs variable selections without LASSO

Works well with nonlinear boundaries

Cons:

Easy to overfit

Requires a lot of fine tuning

Outperformed by other ML techniques

Random Forest

Build many trees using bootstrap samples (bagging)

When training each tree, at each node we take a sample of predictors and split

---

Bootstrapping

Confidence Interval

Finding Confidence Interval:

1. Sort collection in increasing order
2. Find p% (percentile) of n: $(p/100) \times n = k$
3. If k is an integer, take the kth element of the sorted collection
4. Else, round it up to the next integer, and take that element of the sorted collection.

Interpretation of CI

Redraw k sets of bootstrap samples, and calculate k α% Confidence Intervals, α% of them will contain the true parameter of the **sampling population**.

Big Data

Data Organization

Database Data Warehouse:

Data are extracted, transformed, and loaded

Star schema: Fact table: contains **unique** primary keys + foreign key references to dimension tables

Dimension tables: primary keys are unique values to this dimension

Snowflake schema expands on each star to have dependent dimension tables and fact table.

Data Lake - Unstructured Data: Store data in **raw form**

Requires knowledge in access/query

Distributed Computing

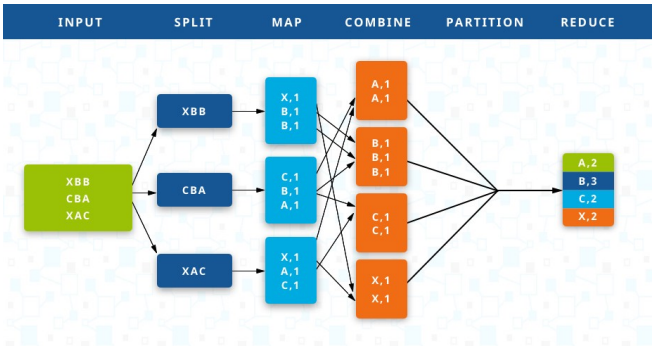
Split one large file into parts, then distribute across multiple machines:

Total fragments made = # of fragments * # of replications

Fragment/machine = Total fragments made / # of machines

Maximum # machine fail = # of replications - 1

Map Reduce Model



Map: takes a dataset and a function and applies function on all points

Reduce: aggregates output of the map stage

*During reduce, all records associated to a given key are sent to the same machine.

*Reading from a large file from distributed machine is faster.

Ray

Distributed execution engine, utilizes multiple CPU:

Worker: worker processes execute tasks

Object Store: immutable objects of processed task stored here, allow workers to efficiently share objects on the same node

Task: stateless function that can be execute on a remote server (functions)

Actor: stateful object that lives in a remote process (class/object-oriented)

*remote process returns object id, use ray.get() to get the actual values back

---