CPSC 340: Machine Learning and Data Mining

Fundamentals continued; k-nearest neighbours

Bonus slides

Decision Theory Discussion

- In other applications, the costs could be different.
  - In cancer screening, maybe false positives are ok, but don't want to have false negatives.
- Decision theory and "darts":
  - http://www.datagenetics.com/blog/january12012/index.html
- Decision theory and video poker:
  - http://datagenetics.com/blog/july32019/index.html
- Decision theory can help with "unbalanced" class labels:
  - If 99% of e-mails are spam, you get 99% accuracy by always predicting "spam".
  - Decision theory approach avoids this.
  - See also precision/recall curves and ROC curves in the bonus material.

Decision Theory and Basketball

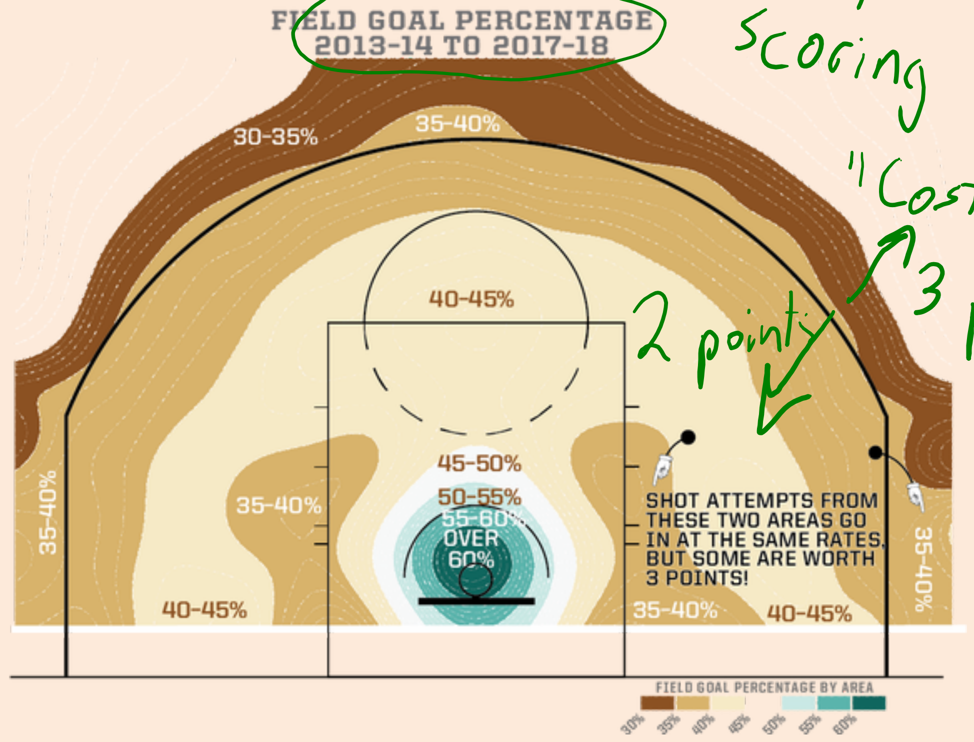
- “How Mapping Shots In The NBA Changed It Forever”

Probability of scoring

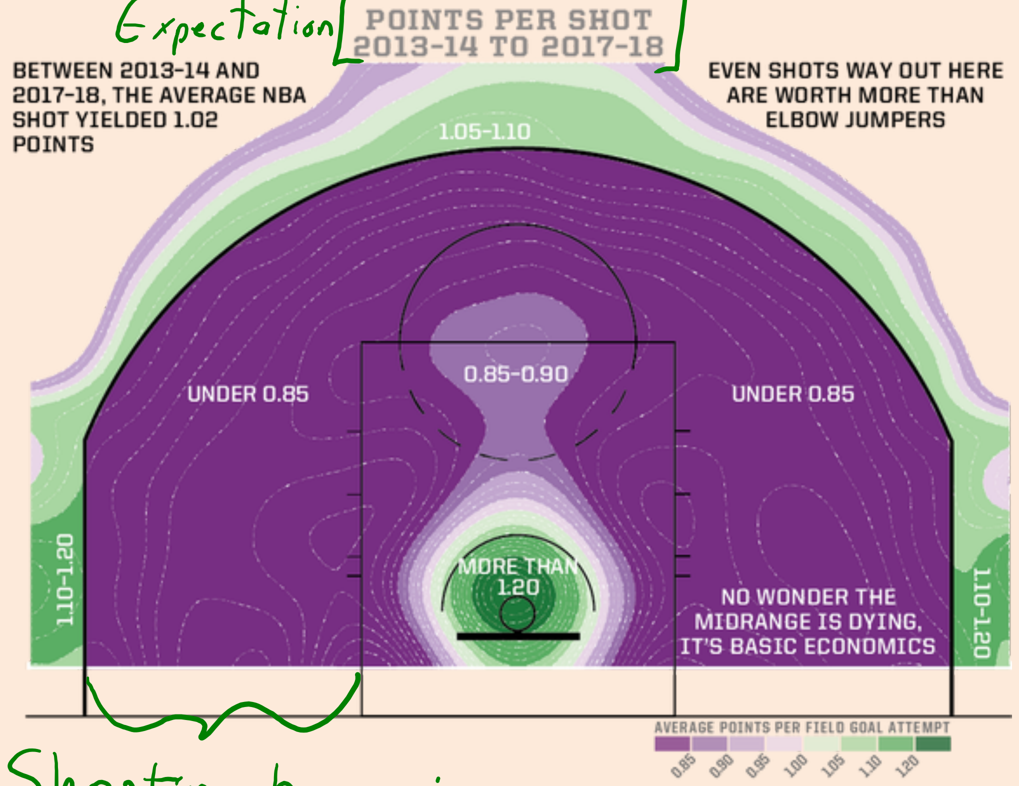
"Cost"

2 points

3 points



Expectation



Shooting here is a bad decision

https://fivethirtyeight.com/features/how-mapping-shots-in-the-nba-changed-it-forever/

Consistency of KNN ('n' going to '∞')

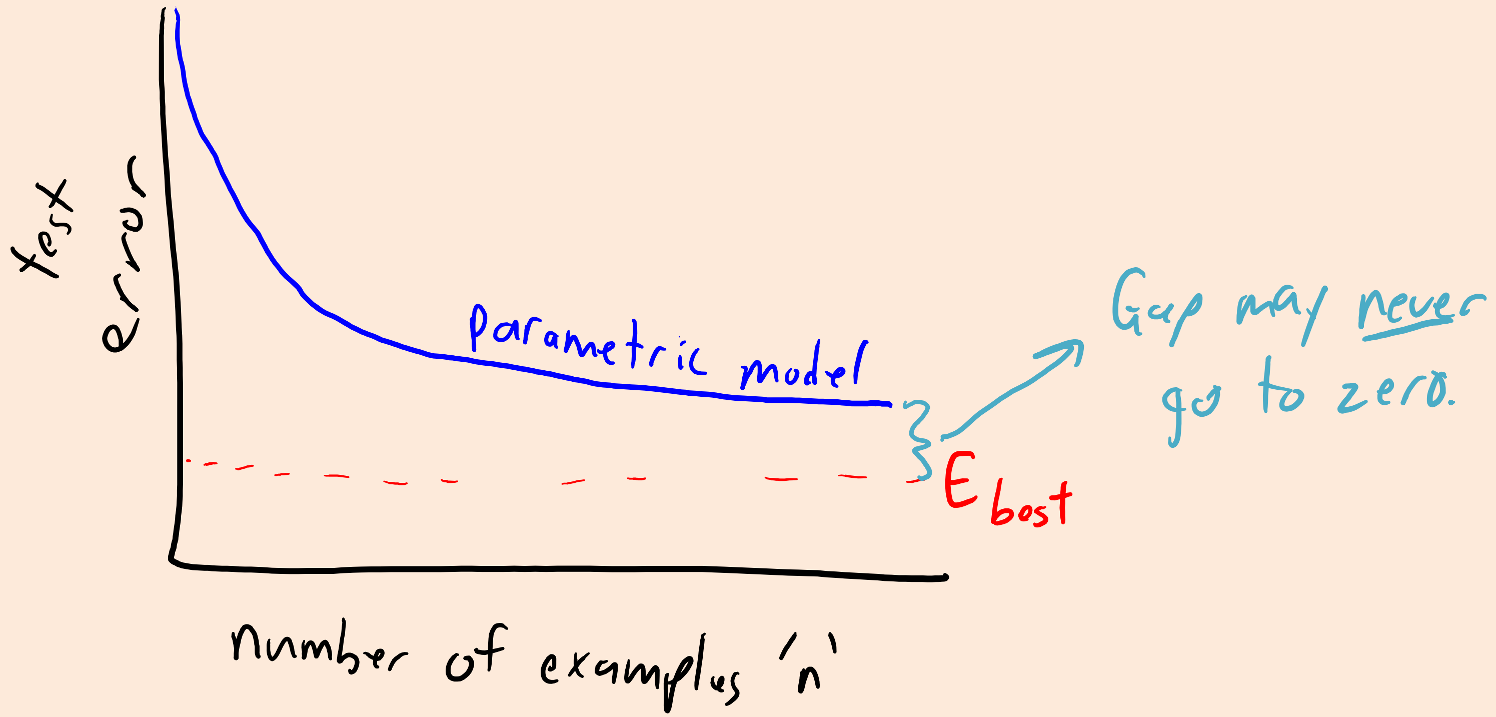
- KNN has appealing consistency properties:
  - As 'n' goes to ∞, KNN test error is less than twice best possible error.
    - For fixed 'k' and binary labels (under mild assumptions).

- Stone's Theorem: KNN is "universally consistent".
  - If k/n goes to zero and 'k' goes to ∞, converges to the best possible error.
    - For example, k = log(n).
    - First algorithm shown to have this property.

- Does Stone's Theorem violate the no free lunch theorem?
  - No: it requires a continuity assumption on the labels.
  - Consistency says nothing about finite 'n' (see "Dont Trust Asymptotics").

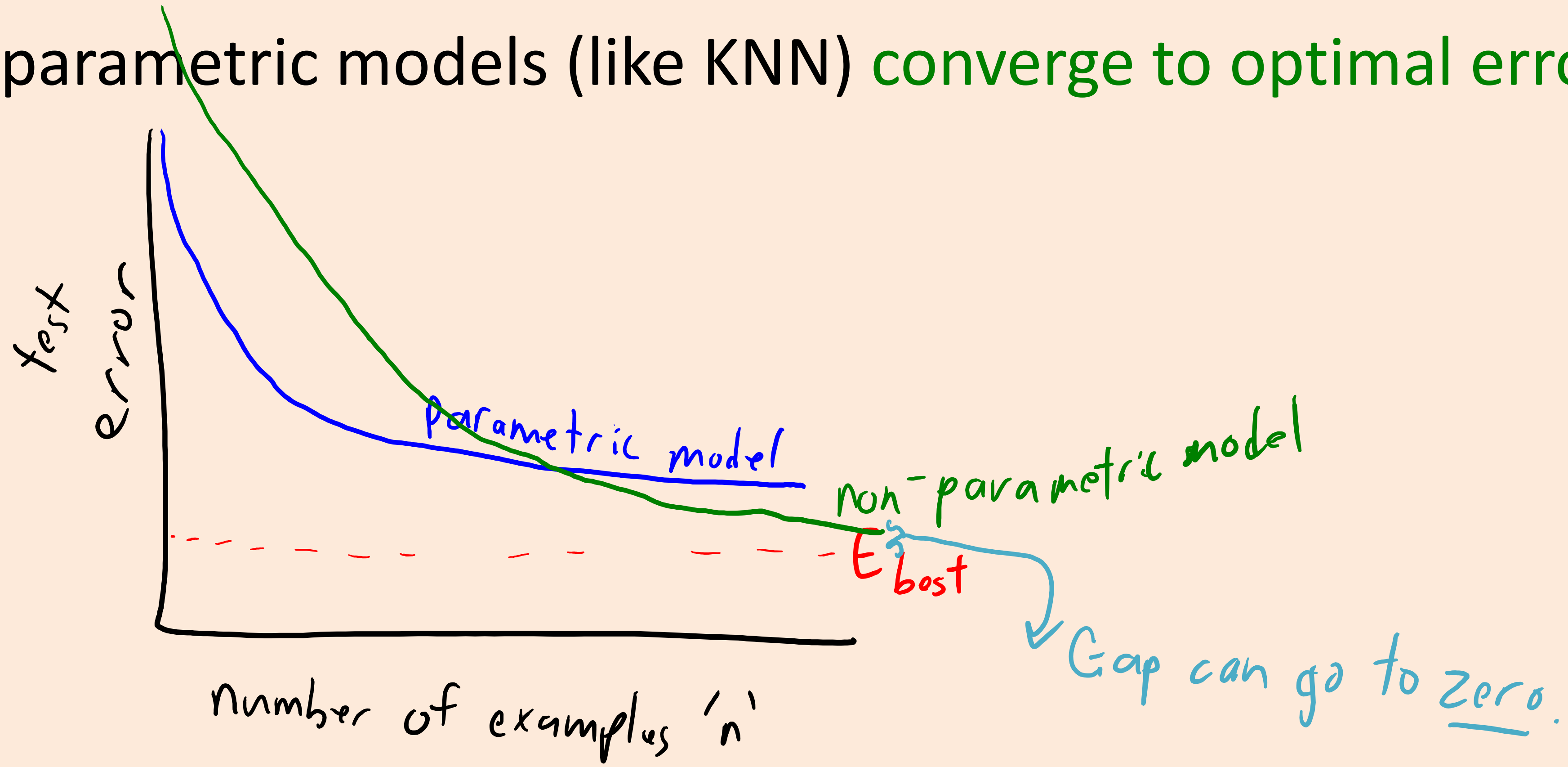
Parametric vs. Non-Parametric Models

- With parametric models, there is an accuracy limit.
  - Even with infinite 'n', may not be able to achieve optimal error (E_{best}).



Parametric vs. Non-Parametric Models

- With parametric models, there is an accuracy limit.
  - Even with infinite 'n', may not be able to achieve optimal error (E_{best}).
- Many non-parametric models (like KNN) converge to optimal error.



More on Weirdness of High Dimensions

- In high dimensions:
  - Distances become less meaningful:
    - All vectors may have similar distances.
  - Emergence of “hubs” (even with random data):
    - Some datapoints are neighbours to many more points than average.
  - Visualizing high dimensions and sphere-packing

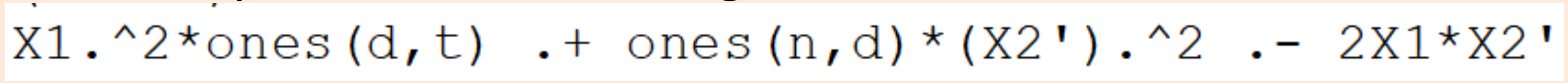
Vectorized Distance Calculation

- To classify ‘t’ test examples based on KNN, cost is O(ndt).
  - Need to compare ‘n’ training examples to ‘t’ test examples, and computing a distance between two examples costs O(d).

- You can do this slightly faster using fast matrix multiplication:
  - Let D be a matrix such that D_{ij} contains:

    $$\|x_i - x_j\|^2 = \|x_i\|^2 - 2x_i^T x_j + \|x_j\|^2$$

    where ‘i’ is a training example and ‘j’ is a test example.
  - We can compute D in Julia using:

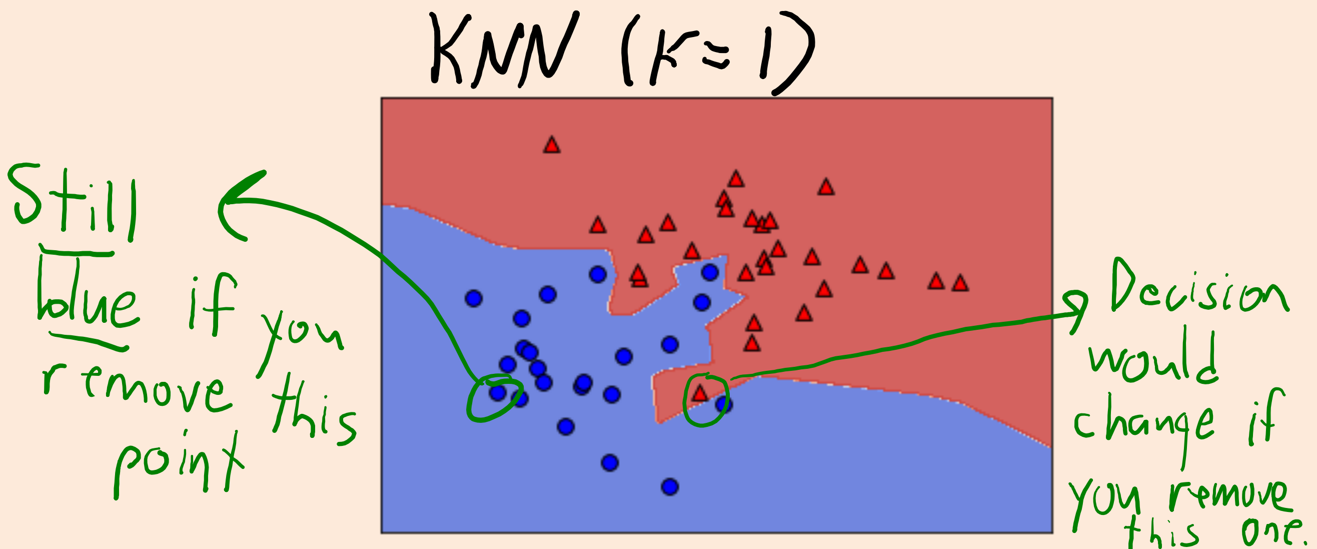


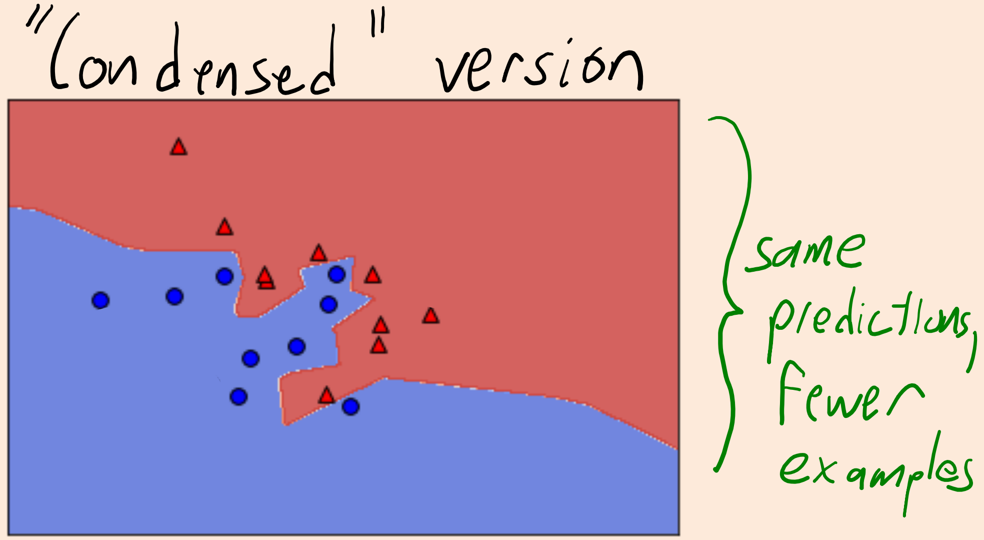
```
X1.^2*ones(d,t) .+ ones(n,d)*(X2').^2 .- 2X1*X2'
```

  - And you get an extra boost because Julia uses multiple cores.

Condensed Nearest Neighbours

- Disadvantage of KNN is slow prediction time (depending on ‘n’).
- Condensed nearest neighbours:
  - Identify a set of ‘m’ “prototype” training examples.
  - Make predictions by using these “prototypes” as the training data.
- Reduces runtime from O(nd) down to O(md).





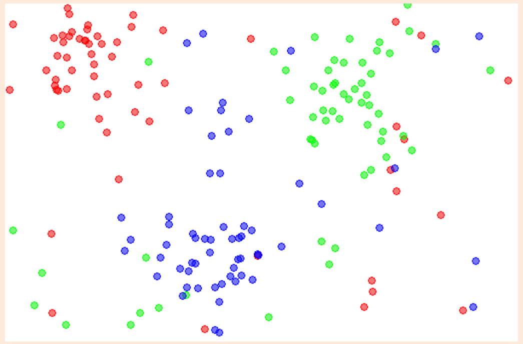
Condensed Nearest Neighbours

- Classic condensed nearest neighbours:
  - Start with no examples among prototypes.
  - Loop through the non-prototype examples 'i' in some order:
    - Classify x_i based on the current prototypes.
    - If prediction is not the true y_i, add it to the prototypes.
  - Repeat the above loop until all examples are classified correctly.
- Some variants first remove points from the original data, if a full-data KNN classifier classifies them incorrectly ("outliers').

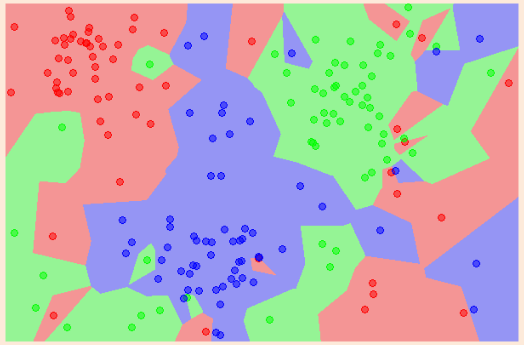
Condensed Nearest Neighbours

- Classic condensed nearest neighbours:

Data:



1NN



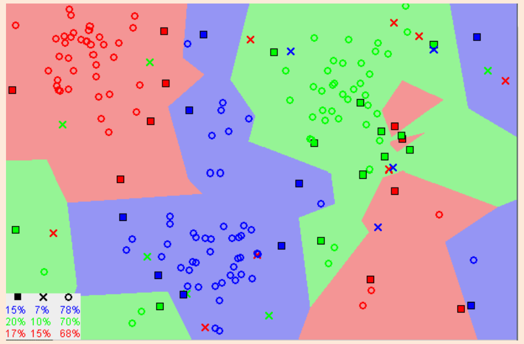
CNN:

□: prototype

○: removed

X: outlier

(using 3NN)



- Recent work shows that finding optimal compression is NP-hard.
  - An approximation algorithm algorithm was published in 2018:
    - "Near optimal sample compression for nearest neighbors"

Back to Decision Trees

- Instead of validation set, you can use CV to select tree depth.

- But you can also use these to decide whether to split:
  - Don’t split if validation/CV error doesn’t improve.
  - Different parts of the tree will have different depths.
- Or fit deep decision tree and use [cross-]validation to prune:
  - Remove leaf nodes that don’t improve CV error.

- Popular implementations that have these tricks and others.

Random Subsamples

- Instead of splitting into k-folds, consider "random subsample" method:
  - At each "round", choose a random set of size 'm'.
    - Train on all examples except these 'm' examples.
    - Compute validation error on these 'm' examples.

- Advantages:
  - Still an unbiased estimator of error.
  - Number of "rounds" does not need to be related to "n".
- Disadvantage:
  - Examples that are sampled more often get more "weight".

Cross-Validation Theory

- Does CV give unbiased estimate of test error?
  - Yes!
    - Since each data point is only used once in validation, expected validation error on each data point is test error.
  - But again, if you use CV to select among models then it is no longer unbiased.

- What about variance of CV?
  - Hard to characterize.
  - CV variance on ‘n’ data points is worse than with a validation set of size ‘n’.
    - But we believe it is close.

- Does cross-validation remove optimization bias?
  - No, but the bias might be smaller since you have more “test” points.

Handling Data Sparsity

- Do we need to store the full bag of words 0/1 variables?
  - No: only need list of non-zero features for each e-mail.

$	Hi	CPSC	340	Vicodin	Offer	…
1	1	0	0	1	0	…
0	0	0	0	1	1	…
0	1	1	1	0	0	…
1	1	0	0	0	1	…

vs.

Non-Zeroes
{1,2,5,…}
{5,6,…}
{2,3,4,…}
{1,2,6,…}

  - Math/model doesn't change, but more efficient storage.

Proof of No Free Lunch Theorem

- Let's show the "no free lunch" theorem in a simple setting:
  - The x^i and y^i are binary, and y^i being a deterministic function of x^i.
- With 'd' features, each "learning problem" is a map from each of the 2^d feature combinations to 0 or 1: $\{0,1\}^d \rightarrow \{0,1\}$

Feature 1	Feature 2	Feature 3
0	0	0
0	0	1
0	1	0
…	…	…

Map 1	Map 2	Map 3	…
0	1	0	…
0	0	1	…
0	0	0	…
…	…	…	…

- Let's pick one of these maps ("learning problems") and:
  - Generate a set training set of 'n' IID samples.
  - Fit model A (convolutional neural network) and model B (naïve Bayes).

Proof of No Free Lunch Theorem

- Define the "unseen" examples as the $(2^d - n)$ not seen in training.
  - Assuming no repetitions of x^i values, and $n < 2^d$.
  - Generalization error is the average error on these "unseen" examples.
- Suppose that model A got 1% error and model B got 60% error.
  - We want to show model B beats model A on another "learning problem".

- Among our set of "learning problems" find the one where:
  - The labels y^i agree on all training examples.
  - The labels y_i disagree on all "unseen" examples.
- On this other "learning problem":
  - Model A gets 99% error and model B gets 40% error.

Proof of No Free Lunch Theorem

- Further, across all “learning problems” with these ‘n’ examples:
  - Average generalization error of **every** model is 50% on unseen examples.
    - It’s right on each unseen example in exactly half the learning problems.
  - With ‘k’ classes, the average error is (k-1)/k (random guessing).

- This is kind of depressing:
  - For general problems, no “machine learning” is better than “predict 0”.

- But the proof also reveals the problem with the NFL theorem:
  - Assumes every “learning problem” is equally likely.
  - World encourages patterns like “similar features implies similar labels”.