k-NN
+
Model Selection
+
Perceptron
**Q:** Why don’t my entropy calculations match those on the slides?

**A:** $H(Y)$ is conventionally reported in “bits” and computed using log base 2. e.g., $H(Y) = - P(Y=0) \log_2 P(Y=0) - P(Y=1) \log_2 P(Y=1)$

**Q:** Why is entropy based on a sum of $p(.) \log p(.)$ terms?

**A:** We don’t have time for a full treatment of why it *has* to be this, but we can develop the right intuition with a few examples...
<table>
<thead>
<tr>
<th>Q:</th>
<th>How do we define a distance function when the features are categorical (e.g. weather takes values {sunny, rainy, overcast})?</th>
</tr>
</thead>
</table>
| A:  | Step 1: Convert from categorical attributes to numeric features (e.g. binary)  
Step 2: Select an appropriate distance function (e.g. Hamming distance) |
Reminders

• Homework 2: Decision Trees
  – Out: Wed, Feb. 10
  – Due: Mon, Feb. 22 at 11:59pm

• Today’s Poll:
  – http://poll.mlcourse.org
  – fill out first two questions about HW1
Moss Cheat Checker
What is Moss?

• Moss (Measure Of Software Similarity): is an automatic system for determining the similarity of programs. To date, the main application of Moss has been in detecting plagiarism in programming classes.

• Moss reports:
  – The Andrew IDs associated with the file submissions
  – The number of lines matched
  – The percent lines matched
  – Color coded submissions where similarities are found
What is Moss?

At first glance, the submissions may look different
What is Moss?
Moss can quickly find the similarities

```python
# Python program to find ordered words
import requests
# Extract the words from the URL below and stores
\[
\text{def getWords():}
\]
\[
\text{# contains about 3500 words}
url = \"https://www.guionera.org/pub/wordlists/uncommon.txt\"
response = requests.get(url)
# Extracts the content of the webpage
wordlist = response.content
# Decodes the UTF-8 encoded text and splits the
# string to turn it into a list of words
wordlist = wordlist.decode(\"utf-8\\").split()
\]
\[
\text{return wordlist}
\]

# Function to determine whether a word is ordered or not
\[
\text{def isOrdered():}
\]
\[
\text{# Getting the wordlist}
wordlist = getWords()
# Since the first few of the elements of the
# dictionary are numbers, getting rid of those
# elements by aliasing adds the first if elements
\]
\[
\text{def getOrdered():}
\]
\[
\text{\# for word in wordlist:
result = Word is ordered
i = 0
l = len(word) - 1
if (l < 3): \# skips the 1 and 2 lettered strings
continue
\]
\[
\text{# for i = l downto 1:
if (isOrdered(i)): ordered = True
result = Word is ordered
break
\]
\]
```

```python
\[
\text{if result = True:}
print(word, \"Ordered\")
\]
```

```python
\[
\text{# Function isOrdered():}
\]
\[
\text{\# if name == \"main\":}
\]
```
DECISION BOUNDARIES
**Dataset:** Outputs \{+, -\}; Features \(x_1\) and \(x_2\)

**Decision Boundary Example**

**In-Class Exercise**

**Question:**
A. Can a **k-Nearest Neighbor classifier** with \(k=1\) achieve **zero training error** on this dataset?
B. If ‘Yes’, draw the learned decision boundary. If ‘No’, why not?

**Question:**
A. Can a **Decision Tree classifier** achieve **zero training error** on this dataset?
B. If ‘Yes’, draw the learned decision boundary. If ‘No’, why not?
k-Nearest Neighbors

*Whiteboard:*

– Decision Tree boundary with continuous features
KNN ON FISHER IRIS DATA
<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal Length</th>
<th>Sepal Width</th>
<th>Petal Length</th>
<th>Petal Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.3</td>
<td>3.0</td>
<td>1.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>4.9</td>
<td>3.6</td>
<td>1.4</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>5.3</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>2.4</td>
<td>3.3</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>5.7</td>
<td>2.8</td>
<td>4.1</td>
<td>1.3</td>
</tr>
<tr>
<td>1</td>
<td>6.3</td>
<td>3.3</td>
<td>4.7</td>
<td>1.6</td>
</tr>
<tr>
<td>1</td>
<td>6.7</td>
<td>3.0</td>
<td>5.0</td>
<td>1.7</td>
</tr>
</tbody>
</table>
Fisher Iris Dataset

Fisher (1936) used 150 measurements of flowers from 3 different species: Iris setosa (0), Iris virginica (1), Iris versicolor (2) collected by Anderson (1936)

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal Length</th>
<th>Sepal Width</th>
<th>Petal Length</th>
<th>Petal Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.3</td>
<td>3.0</td>
<td>1.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>4.9</td>
<td>3.6</td>
<td>1.4</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>5.3</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>2.4</td>
<td>3.3</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>5.7</td>
<td>2.8</td>
<td>4.1</td>
<td>1.3</td>
</tr>
<tr>
<td>1</td>
<td>6.3</td>
<td>3.3</td>
<td>4.7</td>
<td>1.6</td>
</tr>
<tr>
<td>1</td>
<td>6.7</td>
<td>3.0</td>
<td>5.0</td>
<td>1.7</td>
</tr>
</tbody>
</table>

Full dataset: https://en.wikipedia.org/wiki/Iris_flower_data_set
Fisher Iris Dataset

Fisher (1936) used 150 measurements of flowers from 3 different species: Iris setosa (0), Iris virginica (1), Iris versicolor (2) collected by Anderson (1936)

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal Length</th>
<th>Sepal Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.3</td>
<td>3.0</td>
</tr>
<tr>
<td>0</td>
<td>4.9</td>
<td>3.6</td>
</tr>
<tr>
<td>0</td>
<td>5.3</td>
<td>3.7</td>
</tr>
<tr>
<td>1</td>
<td>4.9</td>
<td>2.4</td>
</tr>
<tr>
<td>1</td>
<td>5.7</td>
<td>2.8</td>
</tr>
<tr>
<td>1</td>
<td>6.3</td>
<td>3.3</td>
</tr>
<tr>
<td>1</td>
<td>6.7</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Deleted two of the four features, so that input space is 2D

Full dataset: https://en.wikipedia.org/wiki/Iris_flower_data_set
KNN on Fisher Iris Data

- sepal width
- sepal length

- $y=0$
- $y=1$
- $y=2$
KNN on Fisher Iris Data

Special Case: Nearest Neighbor

3-Class classification ($k = 1$, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification ($k = 2$, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 3, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 4, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 5, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 10, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 20, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 30, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 40, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 50, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification ($k = 60$, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification ($k = 70$, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 80, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 90, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification ($k = 100$, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 110, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 120, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 130, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 140, weights = 'uniform')
KNN on Fisher Iris Data

3-Class classification (k = 140, weights = 'uniform')
KNN on Fisher Iris Data

Special Case: Majority Vote

3-Class classification (k = 150, weights = 'uniform')
KNN ON GAUSSIAN DATA
KNN on Gaussian Data
KNN on Gaussian Data

Classification with KNN (k = 1, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN \((k = 2, \text{weights} = \text{uniform'})\)
KNN on Gaussian Data

Classification with KNN (k = 3, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 4, weights = 'uniform')
KNN on Gaussian Data
KNN on Gaussian Data

Classification with KNN ($k = 9$, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 25, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 36, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 49, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 64, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 81, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 100, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 121, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (\(k = 144\), weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 169, weights = 'uniform')
KNN on Gaussian Data
KNN on Gaussian Data

Classification with KNN (k = 225, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN ($k = 256$, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 289, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 400, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 529, weights = 'uniform')
KNN on Gaussian Data

Classification with KNN (k = 576, weights = 'uniform')
K-NEAREST NEIGHBORS
Questions

• How could k-Nearest Neighbors (KNN) be applied to regression?
• Can we do better than majority vote? (e.g. distance-weighted KNN)
• Where does the Cover & Hart (1967) Bayes error rate bound come from?
KNN Learning Objectives

You should be able to...

• Describe a dataset as points in a high dimensional space [CIML]
• Implement k-Nearest Neighbors with O(N) prediction
• Describe the inductive bias of a k-NN classifier and relate it to feature scale [a la. CIML]
• Sketch the decision boundary for a learning algorithm (compare k-NN and DT)
• State Cover & Hart (1967)'s large sample analysis of a nearest neighbor classifier
• Invent "new" k-NN learning algorithms capable of dealing with even k
• Explain computational and geometric examples of the curse of dimensionality
MODEL SELECTION
WARNING:

• In some sense, our discussion of model selection is premature.
• The models we have considered thus far are fairly simple.
• The models and the many decisions available to the data scientist wielding them will grow to be much more complex than what we’ve seen so far.
Model Selection

**Statistics**

- *Def*: a **model** defines the data generation process (i.e. a set or family of parametric probability distributions)

- *Def*: **model parameters** are the values that give rise to a particular probability distribution in the model family

- *Def*: **learning** (aka. estimation) is the process of finding the parameters that best fit the data

- *Def*: **hyperparameters** are the parameters of a prior distribution over parameters

**Machine Learning**

- *Def*: (loosely) a **model** defines the hypothesis space over which learning performs its search

- *Def*: **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis

- *Def*: the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)

- *Def*: **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select
MODEL SELECTION
Model Selection

WARNING:

• In some sense, our discussion of model selection is premature.
• The models we have considered thus far are fairly simple.
• The models and the many decisions available to the data scientist wielding them will grow to be much more complex than what we’ve seen so far.
Model Selection

**Statistics**
- *Def*: a **model** defines the data generation process (i.e. a set or family of parametric probability distributions)
- *Def*: **model parameters** are the values that give rise to a particular probability distribution in the model family
- *Def*: **learning** (aka. estimation) is the process of finding the parameters that best fit the data
- *Def*: **hyperparameters** are the parameters of a prior distribution over parameters

**Machine Learning**
- *Def*: (loosely) a **model** defines the hypothesis space over which learning performs its search
- *Def*: **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- *Def*: the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- *Def*: **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select
Model Selection

Example: Decision Tree

- model = set of all possible trees, possibly restricted by some hyperparameters (e.g. max depth)
- parameters = structure of a specific decision tree
- learning algorithm = ID3, CART, etc.
- hyperparameters = max-depth, threshold for splitting criterion, etc.

Machine Learning

- Def: (loosely) a model defines the hypothesis space over which learning performs its search
- Def: model parameters are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- Def: the learning algorithm defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- Def: hyperparameters are the tunable aspects of the model, that the learning algorithm does not select
Model Selection

Example: k-Nearest Neighbors

- **model** = set of all possible nearest neighbors classifiers
- **parameters** = none (KNN is an instance-based or non-parametric method)
- **learning algorithm** = for naïve setting, just storing the data
- **hyperparameters** = \( k \), the number of neighbors to consider

Machine Learning

- **Def**: (loosely) a **model** defines the hypothesis space over which learning performs its search
- **Def**: **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- **Def**: the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- **Def**: **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select
Model Selection

Example: Perceptron
• model = set of all linear separators
• parameters = vector of weights (one for each feature)
• learning algorithm = mistake based updates to the parameters
• hyperparameters = none (unless using some variant such as averaged perceptron)

Machine Learning
• Def: (loosely) a model defines the hypothesis space over which learning performs its search
• Def: model parameters are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
• Def: the learning algorithm defines the data-driven search over the hypothesis space (i.e. search for good parameters)
• Def: hyperparameters are the tunable aspects of the model, that the learning algorithm does not select
Model Selection

**Statistics**

- **Def**: a **model** defines the data generation process (i.e. a set or family of parametric probability distributions)

- **Def**: **model parameters** are the values that give rise to a particular probability distribution in the model family

- **Def**: **learning** (aka. estimation) is the process of finding the parameters that best fit the data

- **Def**: **hyperparameters** are the parameters of a prior distribution over parameters

**Machine Learning**

- **Def**: (loosely) a **model** defines the hypothesis space over which the learning algorithm performs its search

- **Def**: **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis

- **Def**: the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)

- **Def**: **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select

If “learning” is all about picking the best **parameters** how do we pick the best **hyperparameters**?
Model Selection

• Two very similar definitions:
  – *Def:* **model selection** is the process by which we choose the “best” model from among a set of candidates
  – *Def:* **hyperparameter optimization** is the process by which we choose the “best” hyperparameters from among a set of candidates *(could be called a special case of model selection)*

• **Both** assume access to a function capable of measuring the quality of a model

• **Both** are typically done “outside” the main training algorithm --- typically training is treated as a black box
# Experimental Design

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training</strong></td>
<td>• training dataset</td>
<td>• best model parameters</td>
<td>We pick the best model parameters by learning on the training dataset for a fixed set of hyperparameters</td>
</tr>
<tr>
<td></td>
<td>• hyperparameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hyperparameter Optimization</strong></td>
<td>• training dataset</td>
<td>• best hyperparameters</td>
<td>We pick the best hyperparameters by learning on the training data and evaluating error on the validation error</td>
</tr>
<tr>
<td></td>
<td>• validation dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Testing</strong></td>
<td>• test dataset</td>
<td>• test error</td>
<td>We evaluate a hypothesis corresponding to a decision rule with fixed model parameters on a test dataset to obtain test error</td>
</tr>
<tr>
<td></td>
<td>• hypothesis (i.e. fixed model parameters)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Special Cases of k-NN

\(k=1\): Nearest Neighbor

\(k=N\): Majority Vote
Example of Hyperparameter Opt.

**Whiteboard:**

- Special cases of k-Nearest Neighbors
- Choosing k with validation data
- Choosing k with cross-validation
Cross-Validation

Cross validation is a method of estimating loss on held out data

**Input:** training data, learning algorithm, loss function (e.g. 0/1 error)

**Output:** an estimate of loss function on held-out data

**Key idea:** rather than just a single “validation” set, use many!
( Error is more stable. Slower computation. )

\[ D = \begin{bmatrix}
  y^{(1)} & x^{(1)} \\
  y^{(2)} & x^{(2)} \\
  \vdots & \vdots \\
  y^{(N)} & x^{(N)}
\end{bmatrix} \]

**Algorithm:**
Divide data into folds (e.g. 4)
1. Train on folds \{1,2,3\} and predict on \{4\}
2. Train on folds \{1,2,4\} and predict on \{3\}
3. Train on folds \{1,3,4\} and predict on \{2\}
4. Train on folds \{2,3,4\} and predict on \{1\}

Concatenate all the predictions and evaluate loss (almost equivalent to averaging loss over the folds)

**Definition:**

\( N \)-fold cross validation = cross validation with \( N \) folds
## Experimental Design

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training</strong></td>
<td>• training dataset</td>
<td>• best model parameters</td>
</tr>
<tr>
<td></td>
<td>• hyperparameters</td>
<td></td>
</tr>
<tr>
<td><strong>Hyperparameter Optimization</strong></td>
<td>• training dataset</td>
<td>• best hyperparameters</td>
</tr>
<tr>
<td></td>
<td>• validation dataset</td>
<td></td>
</tr>
<tr>
<td><strong>Cross-Validation</strong></td>
<td>• training dataset</td>
<td>• cross-validation error</td>
</tr>
<tr>
<td></td>
<td>• validation dataset</td>
<td></td>
</tr>
<tr>
<td><strong>Testing</strong></td>
<td>• test dataset</td>
<td>• test error</td>
</tr>
<tr>
<td></td>
<td>• hypothesis (i.e. fixed model parameters)</td>
<td></td>
</tr>
</tbody>
</table>

We pick the best model parameters by learning on the training dataset for a fixed set of hyperparameters.

We pick the best hyperparameters by learning on the training data and evaluating error on the validation data.

We estimate the error on held out data by repeatedly training on N-1 folds and predicting on the held-out fold.

We evaluate a hypothesis corresponding to a decision rule with fixed model parameters on a test dataset to obtain test error.
Q: We pick the best hyperparameters by learning on the training data and evaluating error on the validation error. For our final model, should we then learn from training + validation?

A: Yes.

Let's assume that \{train-original\} is the original training data, and \{test\} is the provided test dataset.

1. Split \{train-original\} into \{train-subset\} and \{validation\}.
2. Pick the hyperparameters that when training on \{train-subset\} give the lowest error on \{validation\}. Call these hyperparameters \{best-hyper\}.
3. Retrain a new model using \{best-hyper\} on \{train-original\} = \{train-subset\} \cup \{validation\}.
4. Report test error by evaluating on \{test\}.

Alternatively, you could replace Steps 1-2 with the following:

1. Pick the hyperparameters that give the lowest cross-validation error on \{train-original\}. Call these hyperparameters \{best-hyper\}. 
k-NN: Choosing k

Fisher Iris Data: varying the value of k
k-NN: Choosing k

Gaussian Data: varying the value of k
Model Selection

WARNING (again):
– This section is only scratching the surface!
– Lots of methods for hyperparameter optimization: (to talk about later)
  • Grid search
  • Random search
  • Bayesian optimization
  • Graduate-student descent
  • ...

Main Takeaway:
– Model selection / hyperparameter optimization is just another form of learning
Hyperparameter Optimization

**Setting:** suppose we have hyperparameters $\alpha$, $\beta$, and $\chi$ and we wish to pick the “best” values for each one

**Algorithm 1: Grid Search**

- Pick a set of values for each hyperparameter
  $\alpha \in \{a_1, a_2, ..., a_n\}$, $\beta \in \{b_1, b_2, ..., b_n\}$, and $\chi \in \{c_1, c_2, ..., c_n\}$
- Run a grid search

  for $\alpha \in \{a_1, a_2, ..., a_n\}$:
    for $\beta \in \{b_1, b_2, ..., b_n\}$:
      for $\chi \in \{c_1, c_2, ..., c_n\}$:
        $\theta = \text{train}(D_{\text{train}}; \alpha, \beta, \chi)$
        $\text{error} = \text{predict}(D_{\text{validation}}; \theta)$

- return $\alpha$, $\beta$, and $\chi$ with lowest validation error
Hyperparameter Optimization

**Setting:** suppose we have hyperparameters $\alpha$, $\beta$, and $\chi$ and we wish to pick the “best” values for each one.

**Algorithm 2: Random Search**

- Pick a range of values for each parameter
  $\alpha \in \{a_1, a_2, \ldots, a_n\}$, $\beta \in \{b_1, b_2, \ldots, b_n\}$, and $\chi \in \{c_1, c_2, \ldots, c_n\}$
- Run a random search

  for $t = 1, 2, \ldots, T$:
  sample $\alpha$ uniformly from $\{a_1, a_2, \ldots, a_n\}$
  sample $\beta$ uniformly from $\{b_1, b_2, \ldots, b_n\}$
  sample $\chi$ uniformly from $\{c_1, c_2, \ldots, c_n\}$
  $\theta = \text{train}(D_{\text{train}}; \alpha, \beta, \chi)$
  error = $\text{predict}(D_{\text{validation}}; \theta)$

- return $\alpha$, $\beta$, and $\chi$ with lowest validation error
Question:
True or False: given a finite amount of computation time, grid search is more likely to find good values for hyperparameters than random search.

Answer:
Model Selection Learning Objectives

You should be able to...

• Plan an experiment that uses training, validation, and test datasets to predict the performance of a classifier on unseen data (without cheating)

• Explain the difference between (1) training error, (2) validation error, (3) cross-validation error, (4) test error, and (5) true error

• For a given learning technique, identify the model, learning algorithm, parameters, and hyperparameters

• Define "instance-based learning" or "nonparametric methods"

• Select an appropriate algorithm for optimizing (aka. learning) hyperparameters
THE PERCEPTRON ALGORITHM
Perceptron: History

Imagine you are trying to build a new machine learning technique... your name is Frank Rosenblatt... and the year is 1957
Perceptron: History

Imagine you are trying to build a new machine learning technique... your name is Frank Rosenblatt... and the year is 1957.

The New Yorker, December 6, 1958 P. 44

Talk story about the perceptron, a new electronic brain which hasn't been built, but which has been successfully simulated on the I.B.M. 704. Talk with Dr. Frank Rosenblatt, of the Cornell Aeronautical Laboratory, who is one of the two men who developed the prodigy; the other man is Dr. Marshall C. Yovits, of the Office of Naval Research, in Washington. Dr. Rosenblatt defined the perceptron as the first non-biological object which will achieve an organization of its external environment in a meaningful way. It interacts with its environment, forming concepts that have not been made ready for it by a human agent. If a triangle is held up, the perceptron's eye picks up the image & conveys it along a random succession of lines to the response units, where the image is registered. It can tell the difference between a cat and a dog, although it wouldn't be able to tell whether the dog was to the left or right of the cat. Right now it is of no practical use, Dr. Rosenblatt conceded, but he said that one day it might be useful to send one into outer space to take in impressions for us.
Linear Models for Classification

Key idea: Try to learn this hyperplane directly

Looking ahead:
- We’ll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines

Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]
In-Class Exercise

Draw a picture of the region corresponding to:

\[ w_1 x_1 + w_2 x_2 + b > 0 \]

where \( w_1 = 2, w_2 = 3, b = 6 \)

Draw the vector \( \mathbf{w} = [w_1, w_2] \)