Goals for Today’s Recitation

1. Framing Supervised Machine Learning Problems
2. Discussion of Example Supervised Machine Learning Algorithms
   a. Naive Bayes
   b. Support Vector Machines
   c. Random Forests (& Decision Trees)
3. Generative vs. Discriminative Models
4. Project Questions

Framing Supervised Machine Learning Problems

What makes a problem supervised? As discussed in lecture, a supervised learning problem involves learning from pre-labeled data. In other words, we're asked to predict some ground truth after given data and their respective ground truth labels.

Within supervised learning, two types:

1. Regression
2. Classification

Example supervised problems:
- Predicting the integer value of handwritten digits
- Classifying Tumors with Array Data (looking at gene expression for tumor vs. non-tumor)
- Classifying whether a new protein is mitochondrial based on Targeting signal, protein domains, co-expression, homology, induction, motifs)

In practice, how do we develop a model for X, where X is some predictive task?

1. Feature engineering: we need to generate problem-specific features that we expect to have predictive value over the ground-truth label
2. Training: we split our dataset into training and testing portions. We train our model using the training data
3. Testing: We run our model on our testing data and evaluate its performance

If Training performance >> Testing performance, we may be overfitting our data
If Training performance itself is low, we may be underfitting our data

Common methods of evaluating classification based supervised models:

1. Confusion Matrix: plot out true positives (tp), false positives (fp), true negatives (tn), false negatives (fn)
2. Precision Recall
3. AUC: “Area under the curve”.

Precision: \( \frac{tp}{tp + fp} \)
Recall (specificity) = \(\frac{tp}{tp+fn}\)
Sensitivity: \(\frac{tp}{tp + fn}\)
Specificity: \(\frac{tn}{tn + fp}\)

Example Confusion Matrix:

<table>
<thead>
<tr>
<th></th>
<th>Predicted: NO</th>
<th>Predicted: YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual: NO</td>
<td>TN = 50</td>
<td>FP = 10</td>
</tr>
<tr>
<td>Actual: YES</td>
<td>FN = 5</td>
<td>TP = 100</td>
</tr>
</tbody>
</table>
<pre><code>                   | 55            | 110            |
</code></pre>

Common methods of evaluating regression based supervised models: squared distances from truth label.

**Example ML Algorithm: Naive Bayes**

In recitation 1, we touched on Bayes rule.

\[
P(A|B) = \frac{P(B|A)P(A)}{P(B)}
\]

We can now re-frame bayes rule as the relationship of conditional probabilities between a set of features and the class those features belong to.

\[
P(C|F) = \frac{P(F|C)P(C)}{P(F)}
\]

\[
posterior = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}
\]

Predict the label for each feature set as the class with highest posterior probability. Extends well to multiclass and multilabel. Multiclass: predict classes 1...k, and pick the highest probability. Multilabel: separate classifier for each class.

But say we have 10 features each of which can take 10 different values? There are \(10^{10}\) separate feature vectors, and to perform the calculations above, we would need to know the joint probabilities for all \(10^{10}\) combinations.

This isn't feasible because:
1. We might not have the at least 10 billion data points to be able to calculate \(P(F|C)\) [or we would have a lot of 0s]
2. This isn't computationally feasible to do (tough to store all of these values)

Therefore, we make the **Naive Bayes assumption** that all features are independent.
By the chain rule, we can just multiply individual features

\[
P(C \mid <f_1, f_2, f_\ldots>) = \frac{P(f_1|C)P(f_2|c)\ldots P(C)}{P(f_1)P(f_2)P(f_3)\ldots}
\]

This greatly reduces the computational complexity of the problem from exponential to polynomial time.

Training: compute \(P(F|C)\) through frequency counts
Testing: compute \(P(C|F)\)

**Support Vector Machines**
Given a point in \(\mathbb{R}^n\), output either +1 or -1 (+1 or -1) represent class labels
High level strategy: pick a “hyperplane” that separates points in between the training data of different classes.

How do we choose the line?
- There are many options
- Let’s pick the line that gives us the maximum margin between the two samples

![Diagram of Support Vector Machines](image)

Specifically, we want to minimize \(\|w\|^2/2\) (where \(w\) is the normal vector to the hyperplane separating the two classes above). Minimizing this value gives us the maximum margin (which is of length \(2/\|w\|\)).
A second constraint for this minimization is: \( wx + b \geq 1 \) if the training point \( x \) is has a positive label and \( wx + b \leq -1 \) if \( x \) has a negative label. How do we classify a new point? We look at sign \( (wx + b) \). If it’s greater than 0, then classify as positive, less than 0, classify as negative?

What if we can’t fit a hyperplane across the data because there doesn’t seem to be a hyperplane that can actually separate all datapoints into two classes?

1. Soft margin: allow misclassifications in the training data (fit a hyperplane but allow errors in the minimization \( \rightarrow \min \|w\|^2/2 + c \sum a_i \) where \( a \)'s are slack variables)

2. Kernel trick: map data into a higher dimensional space where a hyperplane can separate the data

![Figure 1: Transforming the data can make it linearly separable](Image)

**Kernel Trick** ([Source](#))

**Decision Trees and Random Forests**

A decision tree is a tree that at each node has a “decision” at each node. In the context of learning, a decision tree, when given a set of features, gives you a series of “if-then” statements to follow to find out what label that feature should be

Example:
Why decision trees?
1. White box, very interpretable
2. Can recover which features are most important by looking at the top of the decision tree
3. Can identify which features might be associated with overfitting at the bottom of the decision tree.

What makes a good decision tree?
1. In as few steps as possible, be able to classify a sample as accurately as possible

Algorithm to obtain a “good” decision tree:
If should_split_sample:
   Pick decision criteria that optimally splits
   Recurse on subtrees

Should split sample: when we still don’t know what label this should be (i.e. all of the decision stubs don’t result in the same label).

Optimal split: minimize the impurity of the partitions
1. Entropy: $\sum_k P(y = k) \log(P(y = k))$
2. Gini coefficient: $1 - \sum_k P(y = k)^2$
(Examples - if everything the same, entropy = 0)
Question: why do decision trees naturally overfit? What are some strategies to get around this? (because they continue until same label tree, often get too deep)

Random Forest is an extension of decision tree classifiers:
1. Multiple trees
2. Shorter trees
What this gives us: more resistance to overfitting

What parameters does a Random Forest have:
1. Number of trees: tradeoff between computational time
2. Number of Samples: \( N \)
3. Number of Features: \( M \)

Random Forest Algorithm:
1. Select number of trees to grow
   a. Choose \( N \) samples with replacement
   b. **At each node**, Choose \( m << M \) feature variables w/o replacement and make all branching decisions based on that. An example is \( m = \sqrt{M} \)
   c. Recurse to b, continue till uniform decisions
2. Testing: given an unlabeled feature, let each tree output its “class” and treat that as a vote. I.e. choose the class with the most votes.

Why do we choose \( N \) samples w/ replacement? **Cross-validation** (training the model on different subsets of the data)

Parameterization:
1. If \( m \) is large: deep, strongly correlated trees -> overfitting
2. If \( m \) is small: weakly correlated trees, high out of bag error. P

**Out of Bag Error**: OOB is the mean prediction error on each training sample \( x_i \), using only the trees that did not have \( x_i \) in their bootstrap sample.
Generative vs. Discriminative Models

Generative
1. Bayesian Classification (e.g. Naïve Bayes)
2. Pose classification problem in prob terms
3. Model feature distribution in different classes – Use probability calculus for making decisions

Discriminative
1. E.g. Support Vector Machines, Decision Trees, RF
2. No modeling of underlying distributions
3. Make decisions using distance from boundaries

*Discriminative models care about the separating the data into classes, but don’t reference an explicit model in doing so.*