CS188, Winter 2017
Problem Set 1: Decision trees
Due Feb 2,2017, 4:00pm

Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

Parts of this assignment are adapted from course material by Andrea Danyluk (Williams), Tom Mitchell and Maria-Florina Balcan (CMU), Stuart Russell (UC Berkeley) and Jessica Wu (Harvey Mudd).

1 Maximum Likelihood Estimation [15 pts]

Suppose we observe the values of n independent random variables $X_1, \ldots, X_n$ drawn from the same Bernoulli distribution with parameter θ[1]. In other words, for each X_i, we know that

$$P(X_i = 1) = \theta \quad \text{and} \quad P(X_i = 0) = 1 - \theta.$$

Our goal is to estimate the value of θ from these observed values of X_1 through X_n.

For any hypothetical value $\hat{\theta}$, we can compute the probability of observing the outcome $X_1, \ldots, X_n$ if the true parameter value θ were equal to $\hat{\theta}$. This probability of the observed data is often called the *likelihood*, and the function $L(\theta)$ that maps each θ to the corresponding likelihood is called the *likelihood function*. A natural way to estimate the unknown parameter θ is to choose the θ that maximizes the likelihood function. Formally,

$$\hat{\theta}_{MLE} = \arg\max_{\theta} L(\theta).$$

(a) Write a formula for the likelihood function, $L(\theta) = P(X_1, \ldots, X_n; \theta)$. Your function should depend on the random variables $X_1, \ldots, X_n$ and the hypothetical parameter θ. Does the likelihood function depend on the order of the random variables?

(b) Since the log function is increasing, the θ that maximizes the *log likelihood* $\ell(\theta) = \log(L(\theta))$ is the same as the θ that maximizes the likelihood. Find $\ell(\theta)$ and its first and second derivatives, and use these to find a closed-form formula for the MLE.

(c) Suppose that $n = 10$ and the data set contains six 1s and four 0s. Write a short program `likelihood.py` that plots the likelihood function of this data for each value of $\hat{\theta}$ in $\{0, 0.01, 0.02, \ldots, 1.0\}$ (use `np.linspace(...)` to generate this spacing). For the plot, the x-axis should be θ and the y-axis $L(\theta)$. Scale your y-axis so that you can see some variation in its value. Include the plot in your writeup. Estimate $\hat{\theta}_{MLE}$ by marking on the x-axis the value of $\hat{\theta}$ that maximizes the likelihood. Does the answer agree with the closed form answer ?

(d) Create three more likelihood plots: one where $n = 5$ and the data set contains three 1s and two 0s; one where $n = 100$ and the data set contains sixty 1s and forty 0s; and one where $n = 10$ and there are five 1s and five 0s. Include these plots in your writeup, and describe how the likelihood functions and maximum likelihood estimates compare for the different data sets.

2 Splitting Heuristic for Decision Trees [14 pts]

Recall that the ID3 algorithm iteratively grows a decision tree from the root downwards. On each iteration, the algorithm replaces one leaf node with an internal node that splits the data based on one decision attribute (or feature). In particular, the ID3 algorithm chooses the split that reduces

[1]This is a common assumption for sampling data. So we will denote this assumption as iid, short for Independent and Identically Distributed, meaning that each random variable has the same distribution and is drawn independent of all the other random variables

the entropy the most, but there are other choices. For example, since our goal in the end is to have the lowest error, why not instead choose the split that reduces error the most? In this problem, we will explore one reason why reducing entropy is a better criterion.

Consider the following simple setting. Let us suppose each example is described by n boolean features: $X = \langle X_1, \ldots, X_n \rangle$, where $X_i \in \{0, 1\}$, and where $n \geq 4$. Furthermore, the target function to be learned is $f : X \to Y$, where $Y = X_1 \vee X_2 \vee X_3$. That is, $Y = 1$ if $X_1 = 1$ or $X_2 = 1$ or $X_3 = 1$, and $Y = 0$ otherwise. Suppose that your training data contains all of the 2^n possible examples, each labeled by f. For example, when $n = 4$, the data set would be

X_1	X_2	X_3	X_4	Y
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	0	1
0	0	1	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	0	1

X_1	X_2	X_3	X_4	Y
0	0	0	1	0
1	0	0	1	1
0	1	0	1	1
1	1	0	1	1
0	0	1	1	1
1	0	1	1	1
0	1	1	1	1
1	1	1	1	1

(a) How many mistakes does the best 1-leaf decision tree make over the 2^n training examples? (The 1-leaf decision tree does not split the data even once. Make sure you answer for the general case when $n \geq 4$.)

(b) Is there a split that reduces the number of mistakes by at least one? (That is, is there a decision tree with 1 internal node with fewer mistakes than your answer to part (a)?) Why or why not?

(c) What is the entropy of the output label Y for the 1-leaf decision tree (no splits at all)?

(d) Is there a split that reduces the entropy of the output Y by a non-zero amount? If so, what is it, and what is the resulting conditional entropy of Y given this split?

3 Entropy and Information [2 pts]

The entropy of a Bernoulli (Boolean 0/1) random variable X with $p(X = 1) = q$ is given by

$$B(q) = -q \log q - (1 - q) \log(1 - q).$$

Suppose that a set S of examples contains p positive examples and n negative examples. The entropy of S is defined as $H(S) = B\left(\frac{p}{p+n}\right)$.

(a) Based on an attribute X_j, we split our examples into disjoint subsets S_k, with p_k positive and n_k negative examples in each. If the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, show that the information gain of this attribute is 0.

4 Programming exercise : Applying decision trees [24 pts]

Submission instructions

- Only provide answers and plots. Do not submit code.

Introduction[2]

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this problem, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Starter Files

code and data

- code : `titanic.py`
- data : `titanic_train.csv`

documentation

- Decision Tree Classifier:
  `http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`
- Cross-Validation:
  `http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html`
- Metrics:
  `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`

Download the code and data sets from the course website. For more information on the data set, see the Kaggle description: `https://www.kaggle.com/c/titanic/data`. (The provided data sets are modified versions of the data available from Kaggle.[3])

[2] This assignment is adapted from the Kaggle Titanic competition, available at `https://www.kaggle.com/c/titanic`. Some parts of the problem are copied verbatim from Kaggle.

[3] Passengers with missing values for any feature have been removed. Also, the categorical feature `Sex` has been mapped to `{'female': 0, 'male': 1}` and `Embarked` to `{'C': 0, 'Q': 1, 'S': 2}`. If you are interested more in this process of *data munging*, Kaggle has an excellent tutorial available at `https://www.kaggle.com/c/titanic/details/getting-started-with-python-ii`.

Note that any portions of the code that you must modify have been indicated with `TODO`. Do not change any code outside of these blocks.

4.1 Visualization [4 pts]

One of the first things to do before trying any formal machine learning technique is to dive into the data. This can include looking for funny values in the data, looking for outliers, looking at the range of feature values, what features seem important, etc.

(a) Run the code (`titanic.py`) to make histograms for each feature, separating the examples by class (e.g. survival). This should produce seven plots, one for each feature, and each plot should have two overlapping histograms, with the color of the histogram indicating the class. For each feature, what trends do you observe in the data?

4.2 Evaluation [20 pts]

Now, let us use `scikit-learn` to train a `DecisionTreeClassifier` on the data.

Using the predictive capabilities of the `scikit-learn` package is very simple. In fact, it can be carried out in three simple steps: initializing the model, fitting it to the training data, and predicting new values.[4]

(b) Before trying out any classifier, it is often useful to establish a *baseline*. We have implemented one simple baseline classifier, `MajorityVoteClassifier`, that always predicts the majority class from the training set. Read through the `MajorityVoteClassifier` and its usage and make sure you understand how it works.

Your goal is to implement and evaluate another baseline classifier, `RandomClassifier`, that predicts a target class according to the distribution of classes in the training data set. For example, if 60% of the examples in the training set have `Survived = 0` and 40% have `Survived = 1`, then, when applied to a test set, `RandomClassifier` should randomly predict 60% of the examples as `Survived = 0` and 40% as `Survived = 1`.

Implement the missing portions of `RandomClassifier` according to the provided specifications. Then train your `RandomClassifier` on the entire training data set, and evaluate its training error. If you implemented everything correctly, you should have an error of 0.485.

(c) Now that we have a baseline, train and evaluate a `DecisionTreeClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Make sure you initialize your classifier with the appropriate parameters; in particular, use the 'entropy' criterion discussed in class. What is the training error of this classifier?

(d) So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let us use cross-validation instead.

[4] Note that almost all of the model techniques in `scikit-learn` share a few common named functions, once they are initialized. You can always find out more about them in the documentation for each model. These are `some-model-name.fit(...)`, `some-model-name.predict(...)`, and `some-model-name.score(...)`.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `train_test_split(...)` from `scikit-learn`. To ensure that we always get the same splits across different runs (and thus can compare the classifier results), set the `random_state` parameter to be the trial number.

Next, use your `error(...)` function to evaluate the training error and (cross-validation) test error of each of your three models. To do this, generate a random 80/20 split of the training data, train each model on the 80% fraction, evaluate the error on either the 80% or the 20% fraction, and repeat this 100 times to get an average result. What are the average training and test error of each of your classifiers on the Titanic data set?

(e) One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let us see whether this is the case for the Titanic data.

One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. Then plot the average training error and test error against the depth limit. (Also plot the average test error for your baseline classifiers. As the baseline classifiers are independent of the depth limit, their plots should be flat lines.) Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot.

(f) Another useful tool for evaluating classifiers is *learning curves*, which show how classifier performance (e.g. error) relates to experience (e.g. amount of training data).

Run another experiment using a decision tree with the best depth limit you found above. This time, vary the amount of training data by starting with splits of 0.05 (5% of the data used for training) and working up to splits of size 0.95 (95% of the data used for training) in increments of 0.05. Then plot the decision tree training and test error against the amount of training data. (Also plot the average test error for your baseline classifiers.) Include this plot in your writeup, and provide a 1-2 sentence description of your observations.