CSM146, Fall 2018
Problem Set 1: Decision trees and k-Nearest Neighbors
Due Oct 29, 2018 at 11:59 pm

Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files and complete the multiple choice questions on CCLE.
- 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), Carlos Guestrin (UW) and Jessica Wu (Harvey Mudd).

1 Splitting Heuristic for Decision Trees [20 pts]

Please check CCLE to answer this question.

2 Entropy and Information [5 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) **(5 pts)** Based on an attribute X_j, we split our examples into k 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.

3 k-Nearest Neighbors and Cross-validation [15 pts]

In the following questions you will consider a k-nearest neighbor classifier using Euclidean distance metric on a binary classification task. We assign the class of the test point to be the class of the majority of the k nearest neighbors. Note that a point can be its own neighbor.

(a) **(5 pts)** What value of k minimizes the training set error for this dataset? What is the resulting training error?

(b) **(5 pts)** Why might using too large values k be bad in this dataset? Why might too small values of k also be bad?

(c) **(5 pts)** What value of k minimizes leave-one-out cross-validation error for this dataset? What is the resulting error?

4 Programming exercise : Applying decision trees and k-nearest neighbors [60 pts]

Submission instructions

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

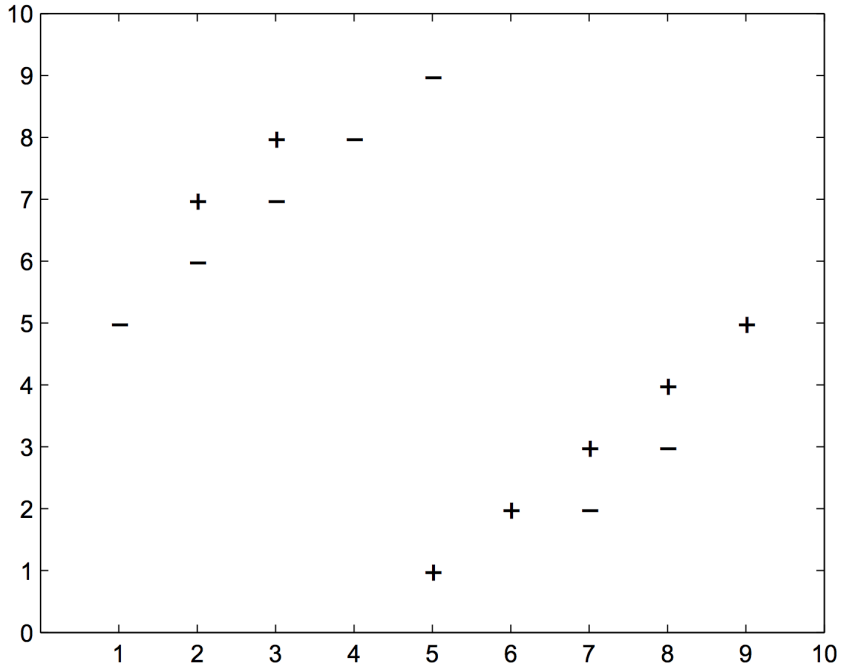

Figure 1: Dataset for KNN binary classification task.

Introduction[1]

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`

[1]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.

- data : `titanic_train.csv`

documentation

- Decision Tree Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- K-Nearest Neighbor Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.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.[2])

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 [5 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) **(5 pts)** 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 [55 pts]

Now, let us use `scikit-learn` to train a `DecisionTreeClassifier` and `KNeighborsClassifier` 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.[3]

(b) **(5 pts)** Before trying out any classifier, it is often useful to establish a *baseline*. We have implemented one simple baseline classifier, `MajorityVoteClassifier`, that always predicts

[2] 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.

[3] 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(...)`.

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) **(5 pts)** 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) **(5 pts)** Similar to the previous question, train and evaluate a `KNeighborsClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Use k=3, 5 and 7 as the number of neighbors and report the training error of this classifier.

(e) **(10 pts)** 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.

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 four models (for the `KNeighborsClassifier`, use k=5). 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?

(f) **(10 pts)** One way to find out the best value of k for `KNeighborsClassifier` is n-fold cross validation. Find out the best value of k using 10-fold cross validation. You may find the `cross_val_score(...)` from `scikit-learn` helpful. Run 10-fold cross validation for all odd numbers ranging from 1 to 50 as the number of neighbors. Then plot the validation error against the number of neighbors, k. Include this plot in your writeup, and provide a 1-2 sentence description of your observations. What is the best value of k?

(g) **(10 pts)** 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. 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.

(h) **(10 pts)** 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). For this experiment, first generate a random 90/10 split of the training data and do the following experiments considering the 90% fraction as training and 10% for testing.

Run experiments for the decision tree and k-nearest neighbors classifier with the best depth limit and k value you found above. This time, vary the amount of training data by starting with splits of 0.10 (10% of the data from 90% fraction) and working up to full size 1.00 (100% of the data from 90% fraction) in increments of 0.10. Then plot the decision tree and k-nearest neighbors training and test error against the amount of training data. Include this plot in your writeup, and provide a 1-2 sentence description of your observations.