CptS 315: Introduction to Data Mining Homework 4 (HW4)

Instructions

- Please use a word processing software (e.g. Microsoft word) to write your answers.
- You will need to submit your answers as a pdf file on Blackboard.
- This assignment is due by the date stated on Blackboard.
- All homeworks should be done individually.

Q1. (20 points) Suppose you are given the following multi-class classification training data, where each input example has three features and output label takes a value from *good*, *bad*, and *ugly*.

- x_1=(0, 1, 0) and y_1=*good*
- x_2=(1, 0, 1) and y_2=*bad*
- x_3=(1, 1, 1) and y_3=*ugly*
- x_4=(1, 0, 0) and y_4=*bad*
- x_5=(0, 0, 1) and y_5=*good*

Suppose we want to learn a linear classifier using multi-class perceptron algorithm and start from the following weights: w_{good}=(0,0,0); w_{bad}=(0,0,0); and w_{ugly}=(0,0,0). Please do hand calculations to show how weights change after processing examples in the same order (i.e., one single pass over the five training examples). See slide 88 of the Perceptron notes.

x_1=(0, 1, 0) and y_1=*good*	*Wbad*	w_{good}=(0,1,0)	w_{bad}=(0,-1,0)	w_{ugly}=(0,0,0)
x_2=(1, 0, 1) and y_2=*bad*	*Wbad*	w_{good}=(0,1,0)	w_{bad}=(0,-1,0)	w_{ugly}=(0,0,0)
x_3=(1, 1, 1) and y_3=*ugly*	*Wgood*	w_{good}=(-1,0,-1)	w_{bad}=(0,-1,0)	w_{ugly}=(1,1,1)
x_4=(1, 0, 0) and y_4=*bad*	*Wugly*	w_{good}=(-1,0,-1)	w_{bad}=(1,-1,0)	w_{ugly}=(0,1,1)
x_5=(0, 0, 1) and y_5=*good*	*Wugly*	w_{good}=(-1,0,0)	w_{bad}=(1,-1,0)	w_{ugly}=(0,1,0)

Q2. (20 points) Suppose you are given the following binary classification training data, where each input example has three features and output label takes a value *good* or *bad*.

- x_1=(0, 1, 0) and y_1=*good*
- x_2=(1, 0, 1) and y_2=*bad*
- x_3=(1, 1, 1) and y_3=*good*
- x_4=(1, 0, 0) and y_4=*bad*
- x_5=(0, 0, 1) and y_5=*good*

Suppose we want to learn a classifier using kernelized perceptron algorithm. Start from the following dual weights: α_1=0; α_2=0; α_3=0; α_4=0; and α_5=0. Please do hand calculations to show how dual weights change after processing examples in the same order (i.e., one single pass over the five training examples). Do this separately for the following kernels:

(a) Linear kernel: $K(x,x^0)=x \cdot x^0$; and

x_1=(0, 1, 0) and y_1=*good*	*0=bad*	*a1+1=1,a2=0,a3=0,a4=0,a5=0*
x_2=(1, 0, 1) and y_2=*bad*	*0=bad*	*a1=1,a2=0,a3=0,a4=0,a5=0*
x_3=(1, 1, 1) and y_3=*good*	*1=good*	*a1=1,a2=0,a3=0,a4=0,a5=0*
x_4=(1, 0, 0) and y_4=*bad*	*0=bad*	*a1=1,a2=0,a3=0,a4=0,a5=0*
x_5=(0, 0, 1) and y_5=*good*	*0=bad*	*a1=1,a2=0,a3=0,a4=0,a5=1*

(b) Polynomial kernel with degree 3: $K(x,x^0)=(x \cdot x^0 + 1)^3$, where $x \cdot x^0$ stands for dot product between two inputs x and x^0.

x_1=(0, 1, 0) and y_1=*good*	*0=bad*	*a1+1=1,a2=0,a3=0,a4=0,a5=0*
x_2=(1, 0, 1) and y_2=*bad*	*1=good*	*a1=1,a2+1=1,a3=0,a4=0,a5=0*
x_3=(1, 1, 1) and y_3=*good*	*35=good*	*a1=1,a2=1,a3=0,a4=0,a5=0*
x_4=(1, 0, 0) and y_4=*bad*	*18=good*	*a1=1,a2=1,a3=0,a4+1=1,a5=0*
x_5=(0, 0, 1) and y_5=*good*	*10=good*	*a1=1,a2=1,a3=0,a4=1,a5=0*

See Algorithm 30 in http://ciml.info/dl/v0_99/ciml-v0_99-ch11.pdf. You can ignore the bias term b.

Q3. (20 points) Suppose $x = (x_1, x_2, \cdots, x_d)$ and $z = (z_1, z_2, \cdots, z_d)$ be any two points in a high-dimensional space (i.e., d is very large). Suppose you are given the following property, where the right-hand side quantity represents the standard Euclidean distance.

$$\left(\frac{1}{\sqrt{d}} \sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}} \sum_{i=1}^{d} z_i \right)^2 \leq \sum_{i=1}^{d} (x_i - z_i)^2 \tag{1}$$

We know that the computation of nearest neighbors is very expensive in the high-dimensional space. Discuss how we can make use of the above property to make the nearest neighbors computation efficient?

One way to get around the high computation cost of finding the nearest neighbor in high dimensional space is to use a scaled means algorithm. This makes it so that instead of finding the nearest neighbor you can instead approximate the nearest neighbor. This is done by hashing similar data into buckets so it is easily accessible when a nearest neighbor is needed for any given data point.

Q4. (20 points) We know that we can convert any decision tree into a set of if-then rules, where there is one rule per leaf node. Suppose you are given a set of rules $R = \{r_1, r_2, , r_k\}$, where r_i corresponds to the i^{th} rule. Is it possible to convert the rule set R into an equivalent decision tree? Explain your construction or give a counterexample.

It is possible to build a decision tree from a rule set using a method called RBDT-1. It generates decisions tress using 3 different criteria that are used to determine the attribute that is best fit to any given node. The 3 atributes are attribute effectiveness, attribute autonomy and minimum value distribution.

Q5. (20 points) Please read the following two papers and write a brief summary of the main points in at most THREE pages.

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Franois Crespo, Dan Dennison: Hidden Technical Debt in Machine Learning Systems. NIPS 2015: 2503-2511 https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems. pdf

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley: The ML test score: A rubric for ML production readiness and technical debt reduction. BigData 2017: 1123-1132 https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/ 46555.pdf

In recent years, a trend has begun to emerge that shows it is easy to build new ML models but increasingly expensive to maintain the models as time passes. This practice is something that occurs in all areas of software engineering and is known as technical debit. In machine learning it has been found that it has a special aptitude for technical debt accumulating at a faster rate than other fields of computer science. In traditional software engineering fields, it has been found that strong abstraction boundaries using encapsulation and modular design help reduce technical debt buildup over time as it allows for easy transitions and changes to a codebase. Machine learning struggles with this concept because of how frequently it is implemented with a strong sense of entanglement. Often in machine learning this happens because ML systems mix signals together, entangling them and making isolation of improvements very difficult or impossible.

In recent years an additional worry during ML development has been reliability, as these systems begin to take on more central roles in real world production settings it has become increasingly critical that these systems are stable. Some ways to ensure the reliability of ML models is to use traditional practices that are used in binary systems in normal software engineering. This includes tests on training data, debuggability, rollbacks and monitoring. One way to ensure that tests can be implemented on training data is to encode intuition about data into a schema so it can be automatically checked before the data is inputted into the model. In addition to the previously mentioned concepts all features that are implemented should undergo privacy checks before they are added to the model to ensure no data has sensitive user data that could be harmful. Only once these features are implemented is it safe to begin looking at adding new features to a system. Upon

the implementation of new features all input code should be tested to ensure bugs in new features are caught before they enter the data generation process where they are harder to find and almost impossible to detect.