STAT 441 Study

Juan Pablo Bello Gonzalez

September 2024

1 1. Learning Concepts

Learning:

- Classification: Categorical outcome
- Regression: Continuous outcome

Supervised: You have labels, ie right answers/confirmation for training data. Easier models than unsupervised. In unsupervised you have to make up distinctive features/categories, you don't even know how many clusters there are.
Interpretation vs. prediction tradeoff: Complex models like trees and neural netoworks are powerful at predicting but there is little interpretability to what the model coefficients actually mean.
You can restrict the class of models to allow for easy interpretation, ie trees with tunned hyperparameters. THEN optimize for prediction wihtin this class models.

1.1 Loss

MSE

1.2 Bias-Variance Tradeoff

MSE loss can be decomposed into three sources of error: **Bias, Variance, and Irreducible error**

$$E[(y_0 - \hat{f}(x_0))^2] = Var(\hat{f}(x_0))) + [Bias(\hat{f}(x_0)))]^2 + Var(\epsilon)$$

All three are non-negarive.

- Variance: How well the model generalizes to other data sets.
- Bias: how well the model fits the training set.

Overfitting: low bias high variance. Perfect fit from highly flexible model, usually with many hyperparameters, means that the model will have high variance and not predict other test sets as well as it predicts that one.
Explain the tradeoff: - Conflict of trying to simulataneously minimize the two sources of error that prevent supervised learning algorithms from generalizing beyond the training set.
MSE is the sum of three things, the graph of MSE makes a U, and the graphs of its three components are

- Irreducible error/Var(epsilon). Constant
- Variance. Increasing with flexibility. The more flexibility, the worse predictive power to data sets outside the training set the model will have.
- Bias. Decaying as flexibility increases. Higher flexibility $\implies$ lower bias $\implies$ better fit.

1.3 Bayes Classifier

A classification algorithm takes in a unit with features (a row on a table) and gives a probability distribution over the categories. Baye's classifier says, classify the unit with the most likely class.

Theorem 1 ***Conditional Bayes Error Rate****: For a given obervation x, is $1 - max_j P(Y = j|x)$ the complement of the Bayes classifier probability.*

Theorem 2 ***Overall Bayes Error Rate****: $1 - E[max_j P(Y = j|x)]$ the EXPECTATION complement of the Bayes classifier probability.*

2 2. Practical aspects

2.1 overfitting

Definition: the model fits the **random noise** of a **sample** rather than the generalizable relationship.
Occurs when the model is **too flexible**. Has too many parameters relative to the number of observations. Advanced learning algorithms are flexible, you don't need a neural network to predict y based on two features, just use some regression.

2.1.1 Defending against overfitting

- Fit to training set
- Evaluate on test set
- split must be chosen AT RANDOM. but fix seed in practice.

By splitting the data into test/train, overfitting can be avoided. If you train on 100% of the data there will be overfitting! We build a model based on the training data but evaluate the model on the test data.

2.2 cross-validation

Split data into k random subsets of equal size, called folds, then fit a model to each of the possible combinations of k-1 folds and evaluate/test on the remaining fold. Average accuracty scores to obtain a good estimate of the true predictive power of the model.
In extreme case: leave-one-out (LOO) cross validation. It's nonsense for most applications.

2.3 Evaluation measures

- Accuracy
- Sensitivity
- Specificity
- Area under the ROC curve
- F

for classification, consider:

- TP
- FP
- RN
- FN

the ordering in the name is (Correct label?, model classification label). Correct label? means that those that start with a true: TP, TN were correctly classified. The seonc instance is what that label classification was duh.
Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

2.3.1 Accuracy, Sensitivty, and Specificity

some formulas for evaluation measures

- Accuracy: (TP+TN)/N, where N is the sum of all four = #observations or sample size. Also Accuracy = $\frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i)$
- Sensitivity (TRUE POS rate): TP/(TP + FN)
- Specificity (TRUE NEG rate): TN/(TN+ FP)

Sensitivity is positive. $1 -$ Specificity is False positive rate.
Ideally we want high specificty and sensitivity.But a trade off needs to be made. The threshold of prediction... you may want to be conservative in your true predictions (high specificity low sensitiviy) if it is for Cancer diagnosis. But liberal if it is about who gets on a survival boat.

2.3.2 ROC

Area under curve is a measure of the sensitivity/specificity tradeoff.

2.3.3 F-measure

$$F = \frac{2 \times TP}{2 \times TP + FP + FN}$$

- Higher values are better.
- appropiate for 0/1 classification.

Important: - Sometimes, depending on the context, you may want to always predict false. An example: when there are ratre or uncommon events. - F measure is affected the most by this. Even if you don't predict false all the time, as long as there is a clear bias in the distribution of false classifications, your TP rate will be very low and so your F-measure will be small as well.

2.3.4 Dealing with rarely occuring categories

Not uncommon to have rare categories, ie. categories that occur infrequently.

- Macro averaging: Treat all categories the same. Default of all algorithms. Default of not doing anything in preprocessing.
- **MICRO** averaging: dominate by frequent categories, giving little weight to rare categories.

2.3.5 One-hot-encoding or "factoring" in R

Sometimes, variates are given in terms of categories themselves like X2 = "Family role" $\in$ [1 : "Father", 2: "son", 3:"sibling"].

We shouldn't just feed these categories as they are to an algorithm because the scale and ordering provided by the category numbers, $[1, 2, 3]$ in the prior example, doesn't mean anything real, and could at best add noise to the data, and at worse confoundingly influence the predictions.

So for a categorical variate with n categories, we take $n - 1$ indicator variables to represent the presence of that category in that variate, this is sometimes known as factoring.

why not do n indicators, one for each category number? Because that would cause multicolinearity. As the $n-$th indicator is just the absence of the $n - 1$ other ones (you can write them in a linear dependence)
NOTE: We often don't leave-one-out in advanced statistical models because they don't always use all variates, or collinearity doesn't affect the model like in regression, where you absoluteley cannot have dependencies between variates.

2.3.6 Variable scaling

Do when variables have vastly differnt ranges, helps give bettwr results by making variates more comparable to each other. Scaling is also called standardization.
Scaling methods:

- Scale them to have mean zero and stdev =1 using $x_{scaled} = \frac{x-\bar{x}}{sd_x}$. Note that indicator variables are not scaled like this, their mean and variance may not be 0,1.
- Scale variables to have the same range.

3 Logistic Regression

Appropiate for 2 class classification $0, 1$.

- $P(y == 1) = p$
- $E(y) = p$

We are using a multilinear regression on all the variates (you can do polynomial thourugh preprocessing by say adding columns x_2^3 to the dataframe)

$$\text{logit}(p) = \log(\frac{p}{1-p}) = X\beta$$

logit tips:

- Recall that $\log(p)$ is defined on $x > 0$. So $\text{logit}(p)$ is only ever defined on $p > 1 - p := p \in (0, 1)$
- logit has Range $= (-\infty, \infty)$ and domain $(0, 1)$, so what it does, implicitly, is take values p from $[0, 1]$ onto the real line.

To **recover** the succcess/label 1 probability, p, we use the **expit** function:

$$expit(x) = \frac{\exp(x)}{1 + exp(x)}$$

$$p = textexpit(\beta_0 + \beta_1 x) = \frac{\exp(\beta_0 + \beta_1 x)}{1 + \exp(\beta_0 + \beta_1 x)}$$

As you would expect, p is equal to a function that takes the linear model onto the range $0, 1$ which is fit for a probability.

3.0.1 Odds

$$odds = \frac{p}{1 - p}$$

log odds is exactly what it sounds.
A coefficient can be interpreted as a log odds ratio of a one unit differnce in the direction of its x variable: If
$logit(p_i) = \beta_0 + \beta_1 x + \cdots + \beta_p x_p$
and $logit(p_j) = \beta_0 + \beta_1(x + 1) + \cdots + \beta_p x_p$ Then $logit(p_j) - logit(p_i) = \beta_1$

$$\beta_1 = log(\frac{odds_j}{odds_i})$$

For interpretation: Exponential coefficients give the log odds ratio of one unit increase in the x variate of the coeficient.
So if $logit(p_{admission}) = 10 + .15x_{average}$
you can say: "on average, each 1% increase in average **increases the odds** of admission by **a factor of** $exp(\beta_1) = exp(0.15)$, with all other variables held constant."
So if the odds $(\frac{p}{1-p})$ of admission for an average of $x_1 = 80$ are, say 7.38, then the revised odds for an average of 81 are $= 7.38 \times exp(.15)$
WE can only make statements about odds. We cannot say anything about how the probabilities p themselves will change when you increase one variate by 1 because the increase is non-constant.

3.0.2 model evaluation

Logistic regression is less prone to overfitting than other learning methods.

- Linearity assumption makes the model highly inflexible.
- There is no tunning parameter by default... you could regularize.
- Usually no test/train/cv split for logistic regression needed.
- it is hard to beat logistic regression (bi class) in terms of prediction.
- it is easier to beat linear regression or multi-class logisti regression.
- Affected a great deal with multicolinearity.
- Only has **linear** decision boundaries

3.0.3 Shaky

the log likelihood maximization and matrix terms.

4 Regularization

- L1: Lasso
- L2: Ridge

4.0.1 Motivation

In regression, OLS is the 'best' (lowest variance) **unbiased** estimator. However, we may be willing to trade a bit of bias for a larger reduction in variance (better prediction)
Recall the bias variance tradeoff: bias and variance cannot be simultaneously minimized.
As model felxibility/complexity/tunnability/# hyperparameters increases:

- Bias decreases: Better fit to the training set.
- Variance increases: Less generalizability to other samples of the population.

For low flexibility (**UNDERFITTING**:) High bias, low variance. For high flexibility, **OVERFITTING**, low bias, high variancce.
Regularization explicitly addresses the tradeoff between overfitting and underfitting. Overfitting occurs when a model is overly flexible, characteristic of highly nonlinear and complex models. However, we also want to avoid underfitting, as using complex models can help capture the true relationships between labels and variables. To balance these opposing goals, regularization allows us to use complex models while adding a penalty for model complexity. This ensures that, among a set of complex models, we select the one that is sufficiently complex without being excessively so.
the new criterion to minimize becomes $MSE + Penalty(\beta)/n = RSS + Penalty(\beta)$

$$\min_{\boldsymbol{\beta}}\{RSS + penalty\}$$

where $RSS = \sum_{i=1}^{n}(y_i - \beta^t x)^2$

4.0.2 Ridge, L2

$$Penalty(\beta) = ||\beta|| = \sqrt{\sum_{i=1}^{p}\beta_i^2}$$

Note, we don't penalize the intercept β_0, i.e. it is not included in the computation of the penalty. You can see this in the fact that the summation in ridge starts at $i = 1$ and not at 0.
For minimization purposes, get rid of the square root to simplify things.

$$\text{Criterion} = \frac{1}{n}(RSS + \sum_{i=1}^{p}\beta_i^2)$$

Now, we introduce a penalty as a function of the coefficients, but we can also include a parameter, known as the **regularization parameter** λ, to adjust its strength. This parameter allows us to fine-tune the penalty, as some datasets may require a stronger penalty than others.

$$\text{CriterionL2} = \frac{1}{n}(RSS + \lambda \sum_{i=1}^{p} \beta_i^2)$$

1. Tunning: estimate lambda via CV (training on multiple sets)
2. Training: find the beta's in the model that minimize the criterion, with the tuned parameter.

Notes:

- The penalty is a function of squared coefficients, so it penalizes larger coefficients more than smaller coefficients.

if you don't want to penalize larger coefficients as strongly, you can use an L1 penalty, which penalizes them lineary. As a bonus, L1 penalty also performs variable selection for ya.

$$\text{CriterionL1} = \frac{1}{n}(RSS + \lambda \sum_{i=1}^{p} |\beta_i|)$$

4.0.3 L1 vs L2

- Usually L2 works better in terms of accuracy/MSE.
- L1 beats L2 when it does variable selection, i.e. when there are a lot of **irrelavant** variables in the data.
- Coefficients in L1 can shrink to zero. Coefficients in L2 do not shrink all the way to zero. Therefore, L1 is used for var selection.
- L1 pernalty is numerically harder to compute, no clean derivatives of absolute values.

Recall that the abstraction of RSS, its generalized criterior to optimize is **likelihood**. We can add penalties to likelihiod to add regularization to generalized models like logistic regression.
For L2 penalty, in general:

$$Criterion = -loglikelihood + \frac{\lambda}{n} \sum_{i=1}^{p} \beta_j^2$$

But in practice, we often scaale the squred coefficients in the penalty by the variance of the variate.

$$Criterion = -loglikelihood + \frac{\lambda}{n} \sum_{i=1}^{p} \beta_j^2 Var(x_j)$$

5 Text/NLP

Initially NLP was hard-code with rules like "if text contains ("keyword1 — keyword2") then ..."
REpresentation of a set of documents as vectors is known as **vector space model**.
We study the use of vector space models to create **ngram-vars** or **bag-of-words**
Single word variables called **unigrams**. We record the frequency.
Now we refine the unigrams:

- We want to further use **stemming** which reduces word conjugations to their root: words "chased", "chases", "chasing" are all considered as the same unigram variable "chase."
- Remove stopwords like "the" "and"

5.0.1 bag-of-words

Ignore word order, just looks at unigram variables defined as the word frequency (or just indicator variables in some applications) for each sentence/text.

If it is important to recognize two word groupings add **bi-grams** : say you want to differentiate if cat chases or dog chases, you would ad bigrams "cat chase" and "dog chase."
In general, you can add whichever **ngram** variables you want which group n words. This is a manual process whih only makes sense if you're working with small sentences or if you know you're working with a specific type of sentence, say with the nouns cat and dog.

You can encode grammar into variables: Begining of line noun, Tony followed by verb, etc.
Bag-of-words eval:

- works well on small to medium texts.
- doesn't work for long texts: too much overlap between words.
- old-school approach, LLMs are SOA.

5.0.2 multicolinearity

for general texts, most words do not appear in every sentence, so most entries in the $x-$matrix are zero.
One solutions is to dimensionality reduction, like PCA. Reduce raw feature space to lower soace with uncorrelated variables.

5.0.3 TF-IDF

Short texts: indicator vars. Long text: frequencies. Issue: diminishing returns of using frequency for classification. Knowing a word occurs is useful, knowing it occurs twice is not twice as useful. So we come up with a better measure for classification.
TF-IDF: Function that dampens the effect of additional words.
Term frequency Let $f_{t,d}$ be the frequency of word t in document d. A document is a sentence or the "text for one obsefvartion."

$$TF = \begin{cases} 1 + log(f_{t,d}) \\ x(n-1) \end{cases}$$

Inverse document frequency: "Inverse" because rare words should be given more weights than common (high frequency) words.

$$IDF = log(N/N_t)$$

Where N is the number of documents. N_t is the number of documents containing word t.
So, **TF-IDF:**

$$TfIdf = TF \times IDF$$

Usually compute TFIDF after stemming.
similarity: To quantify similarity of text documents. A similarity score metric st 0 means documents are dissimilar and 1 they are similar.
Good measure: Cosine of the angle between the vector representations of two documents.

$$cos(\theta) = \frac{x\dot{y}}{|x||y|}$$

6 kNN

Recall: Bayes classifier for new observation $x_0 = argmax_j P(Y = j|X = x_0)$
The idea behind kNN is to estimate it locally.

And indeed. kNN **aims to approximate the Bayes classifier locally.**
for a new observation x_0, find the set of nearest k observations in the training data, call this set N_0. Then your probability estimate for every class just becomes the local probaility in the set of k nearest neighbours:

$$P(Y = j|X = x_0) = \frac{1}{1} \sum_{i \in N_0} I(y_i = j)$$

This is a **memory based algorithm**. No training required, only memory to store all "training set" and then do the "functional lookup" of the knn.

It accomodates **nonlinear** decision boundaries.
In comparisson, logistic regression only has linear dicision boundaries.
The hyperparameter k regulates the bias/variance tradeoff: Small k: low bias (fit) high variance (prediction power.)
Increasing k: increases bias and lowers variance.

6.0.1 ties

break them at random, increase k until tie is broken (easy to do since every prediction is a look up and then computing the set with te k nearest observations) Use 1NN (the nearest observation label.)

6.0.2 metrics

Jaccard Similarity: intersection divided by union of two texts.
Euclidean distance favours 1-word answers.
for k=1 we shuold make no error on the training data but NOT always true. why? because of duplicates. one person may go to pharmacy a and pharmacy b as well. can happen.

6.0.3 Example: Smokers Helpline

Outline:

1. Ngram encoding.
   (a) Unigrams
   (b) Remove stopwords

(c) stemming.

2. Standardize x-vars
3. test/train split using LOO validation for tunning k and metric (euclidean and jacard)

6.0.4 Evaluation

- particularly **sensitive to variable scaling/ standardization**.Eg distance measured in km vs miles may yield vastly differnt results without standardization. because variate measures being differnt means differnt relative distances between points.
- Choose an odd number for k to reduce chances of ties.
- tpically eucledean distance is used but you can use others, or a similarity metric.
- LOO cross validation is appropiate for knn as it is not computationally intensive.
- highly local behaviour.
- good when decision boundary is highly irregular and the training data are sufficiently large.
- highly unstructured, just predicts, doesn't learn anything.

7 Naive Bayes

Conditional densities:

$$f_k(X) = P(X = x|Y = k)$$

probability of observing outcome x in for a given label.
Assumption that, conditional on class k, variables x_j are independent.

$$f_k(X) = \Pi_j^p f_{kj}(x_j)$$

This says (idiot's bayes):

$$P(X = (x_1, x_2, \ldots, x_p)|Y = k) = P(X_1 = x_1|Y = k) \times P(X_2 = x_2|Y = k) \ldots P(X_p = x_p|Y = k)$$

Assumption is always wroten but convenient.
We now just plug in this formula for $f_k(X)$ into the bayes formula to get the naive bayes of the probabilities our classifier aims to maximize $P(Y = y|X = x)$. You can actually ignore the denominator in the naive bayes formula and so the algorithm becomes:

$$argmax_k(\pi_k \prod_{j=1}^{p} P(X_j = x_j|Y = y)) = argmax_k(\pi_k \prod_{j=1}^{p} f_{kj}(x_j))$$

Where there are many variates x_j, multiplying $f_{kj}(x_j)$ for all of them can be result in an **underflow** if all of the probabilities are small, say for the given class x_1 is rarely true.

we take the log to avoid underflow in practice.
Estimate priors from the training data
$\pi_k = n_k/n$
Example: If $P(no|ind, LA, potatoes) \propto d$