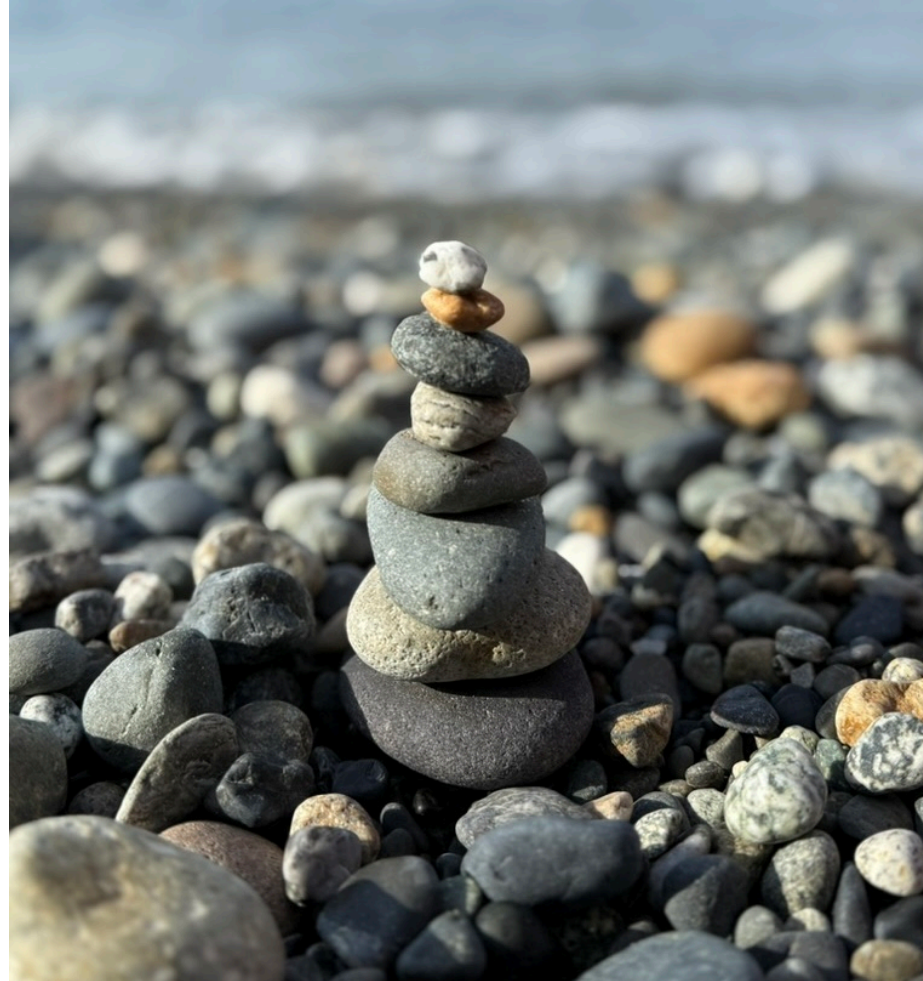


Lecture 7: Linear models

Varada Kolhatkar

Focus on the breath!



Announcements

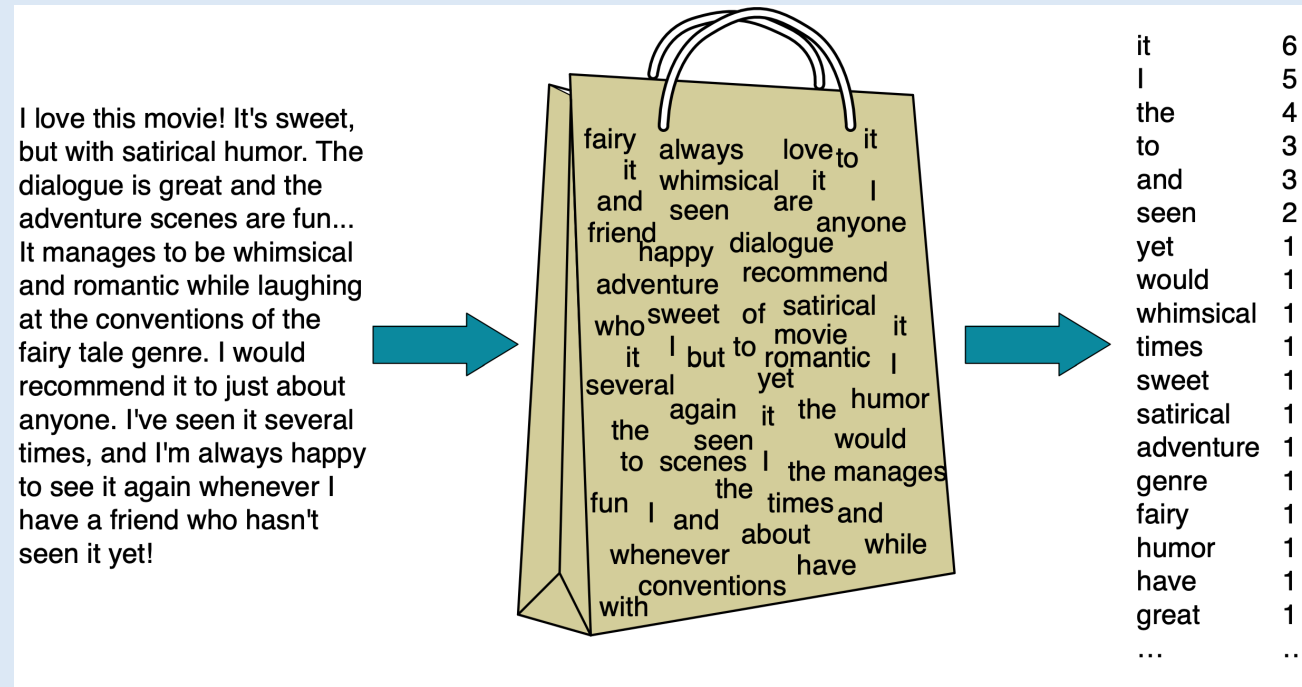
- Important information about midterm 1
- Where to find slides?
 - <https://kvarada.github.io/cpsc330-slides/lecture.html>
- HW3 is due on Monday, Oct 5th, 11:59 pm.

Today's big questions

1. How do linear models turn features into predictions and probabilities?
2. How do loss functions and regularization guide what a model learns?
3. What can coefficients and prediction probabilities tell us and where can they mislead us?

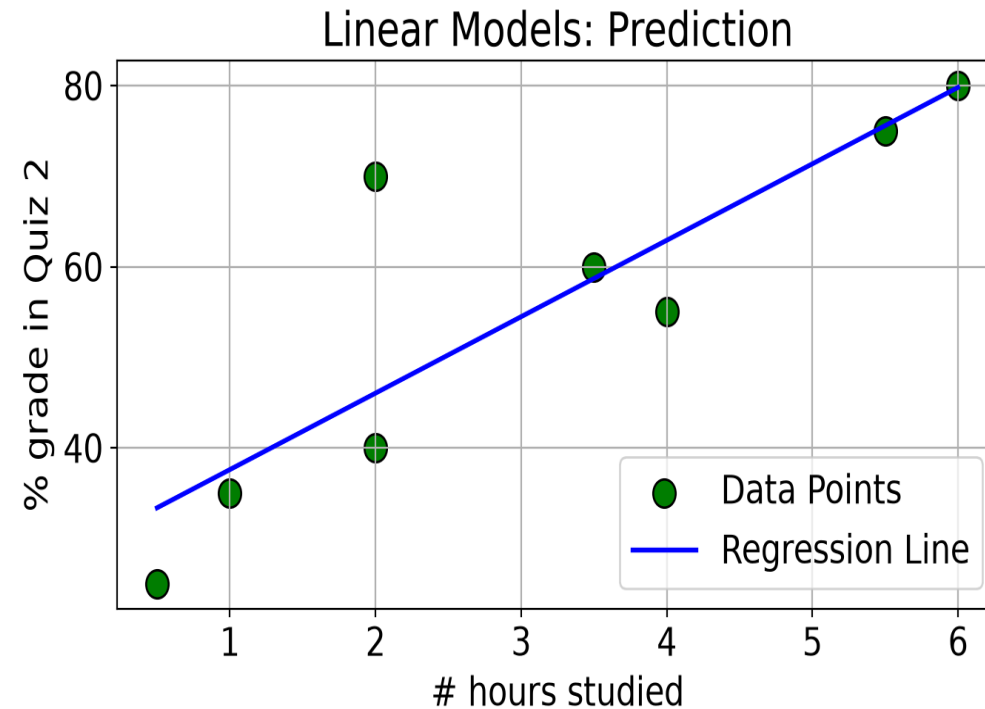
Recap: Dealing with text features

- Preprocessing text to fit into machine learning models using text vectorization.
- Bag of words representation



Linear models

- Linear regression models a numeric target using a weighted sum of the features and an intercept.
- In this case, with only one feature, our model is a straight line.
- What do we need to represent a line?
 - Slope (w_1): Determines the angle of the line.
 - Y-intercept (b): Where the line crosses the y-axis.



- Making predictions: $\hat{y} = w_1 \times \# \text{ hours studied} + b$

Predictions with multiple features

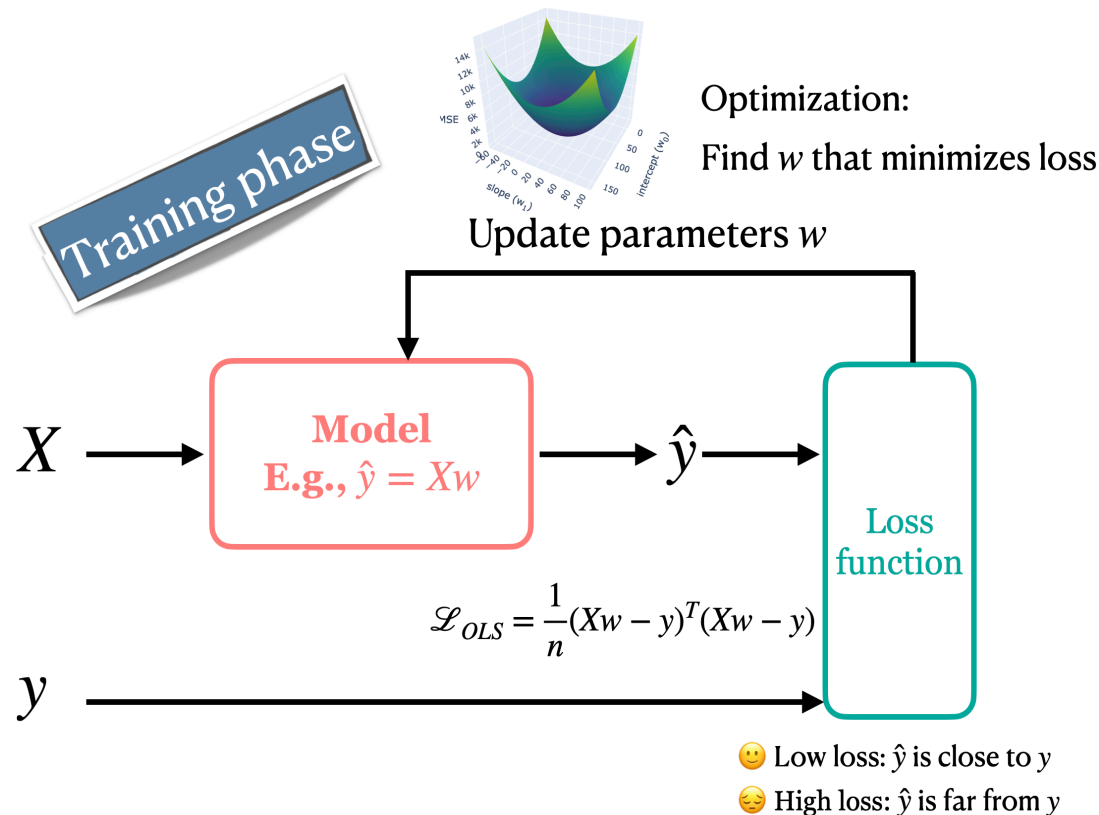
Feature	Value	Coefficient	Contribution
Hours studied (x_1)	5	4	20
Practice sets (x_2)	6	2	12
Intercept (b)			45

The predicted grade is $20 + 12 + 45 = 77$.

$$\hat{y} = w_1x_1 + \dots + w_dx_d + b = \mathbf{w}^T \mathbf{x} + b$$

- The dot product means **multiply corresponding entries and add**.
- We learn one coefficient per feature and an intercept, shared across examples.

What happens during **fit**



Features enter a model; predictions and observed targets enter a loss function; optimization updates the model parameters.

For a linear model, **fit** learns the **coefficients and intercept**:

1. **Predict:** use candidate parameters to compute $\hat{y}$ from the training features.
2. **Measure error:** compare $\hat{y}$ with the observed targets y using a **loss function**.
3. **Optimize:** choose parameters that minimize the fitting objective.

The objective includes the loss and, when used, a **regularization penalty**.

Residuals and squared-error loss

A **residual** is the observed target minus the prediction: $y - \hat{y}$.

Observed grade y	Prediction $\hat{y}$	Residual	Squared error
70	68	2	4
70	66	4	16

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{4 + 16}{2} = 10$$

- Positive residual: prediction too low. Negative residual: prediction too high.
- Ordinary least squares (OLS) chooses coefficients to minimize squared-error loss.
- Squared error penalizes large prediction errors more heavily than small ones.

Your turn: Trace a prediction and its loss

A linear model uses $\hat{y} = 3x + 10$.

Input x	Observed target y	Prediction	Residual	Squared error
2	18	?	?	?
5	20	?	?	?

1. Fill in the missing values and calculate the mean squared error.
2. Which example contributes more to the loss?

Ridge: Loss plus regularization

- `LinearRegression` minimizes squared-error loss.
- Correlated features can lead to large, unstable coefficients.
- `Ridge` adds **regularization**: a penalty that discourages large coefficients.
- Fitting balances agreement with training targets against the penalty.
- Regularization can reduce overfitting; it keeps the same prediction rule.

In this course, we'll use `Ridge` for linear regression.

(Optional) What **Ridge** minimizes

$$\underbrace{\sum_{i=1}^n (y_i - \hat{y}_i)^2}_{\text{fit the training targets}} + \underbrace{\alpha \sum_{j=1}^d w_j^2}_{\text{discourage large coefficients}}$$

- n : training examples; d : features. The intercept is not penalized.
- **alpha** controls the strength of the penalty.
- This uses the **sum** of squared errors. Using the mean changes its balance with the penalty unless **alpha** is adjusted too.

Regularization and scaling

- Larger `alpha` → stronger regularization → smaller coefficients; greater risk of underfitting.
- Smaller `alpha` → weaker regularization; greater risk of overfitting.
- Choose `alpha` using cross-validation on training data; reserve the test set for final evaluation.
- Scale numeric features: their units affect coefficient sizes and hence the penalty.

```
1 ridge_pipe = make_pipeline(StandardScaler(), Ridge(alpha=1.0))  
2 cross_validate(ridge_pipe, X_train, y_train)
```

The pipeline fits the scaler separately within each cross-validation fold.

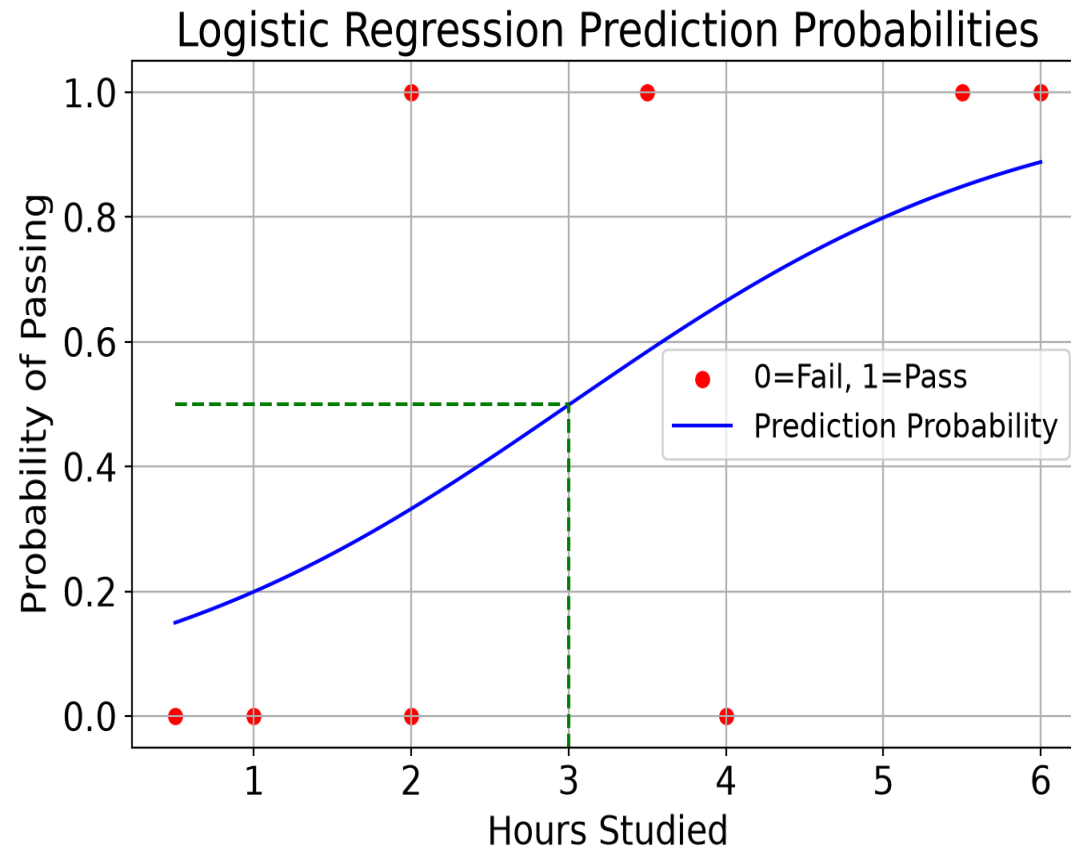
Interpreting regression coefficients

- A coefficient describes the change in the **model's prediction** for a one-unit feature increase, holding other represented features fixed.
- After `StandardScaler`, one unit corresponds to one training-set standard deviation of the original feature.
- Correlated features can share predictive information, making individual coefficients unstable.
- Coefficients describe the fitted model; they do **not** establish causal effects.
- A constant change per unit may miss curves or interactions. Extrapolation can give implausible predictions, such as grades above 100%.

Logistic regression

- Suppose your target is binary: pass or fail
- Logistic regression is used for such binary classification tasks.
- Logistic regression predicts a probability that the given example belongs to a particular class.
- It uses **Sigmoid function** to map any real-valued input into a value between 0 and 1, representing the probability of a specific outcome.
- A threshold (usually 0.5) is applied to the predicted probability to decide the final class label.

Logistic regression: Decision boundary



- Probability of passing:
$$p = \sigma(w^\top x_i + b)$$
$$= \frac{1}{1 + e^{-(w^\top x_i + b)}}$$
- The decision boundary is the point on the x-axis where the corresponding predicted probability on the y-axis is 0.5.

Decision scores, probabilities, and classes

$$z = \mathbf{w}^T \mathbf{x} + b \quad p = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- `decision_function` returns the raw score z .
- `predict_proba` returns probabilities in the order given by `classes_`; p corresponds to `classes_[1]`.
- $z = 0$ corresponds to $p = 0.5$, the default classification boundary.
- The sigmoid is nonlinear, but the **boundary remains linear**: $\mathbf{w}^T \mathbf{x} + b = 0$.
- With two features the boundary is a line; with d features it is a $(d - 1)$ -dimensional hyperplane.

How does **fit** work for
logistic regression?

iClicker: Ranking prediction penalties

We want to classify an image as **cat or dog**. The true class is **cat**.



True class: cat

Image created by GPT-6.1 Sol Light

Model	$P(\text{cat})$	$P(\text{dog})$
A	0.50	0.50
B	0.03	0.97
C	0.90	0.10

Which ranking is **most appropriate** for the penalties these models should receive? Order penalties from **smallest to largest**.

- **A.** $\text{penalty}(\text{A}) < \text{penalty}(\text{B}) < \text{penalty}(\text{C})$
- **B.** $\text{penalty}(\text{C}) < \text{penalty}(\text{A}) < \text{penalty}(\text{B})$
- **C.** $\text{penalty}(\text{B}) < \text{penalty}(\text{A}) < \text{penalty}(\text{C})$
- **D.** All three receive the same penalty.

Log loss

Let p_{true} be the probability assigned to the **true class**.

$$L = -\log(p_{\text{true}})$$

- High probability for the true class $\rightarrow$ small loss.
- Probability near zero for the true class $\rightarrow$ very large loss.
- A **confident incorrect prediction** receives a particularly large penalty.

Here and throughout, $\log$ is the natural logarithm.

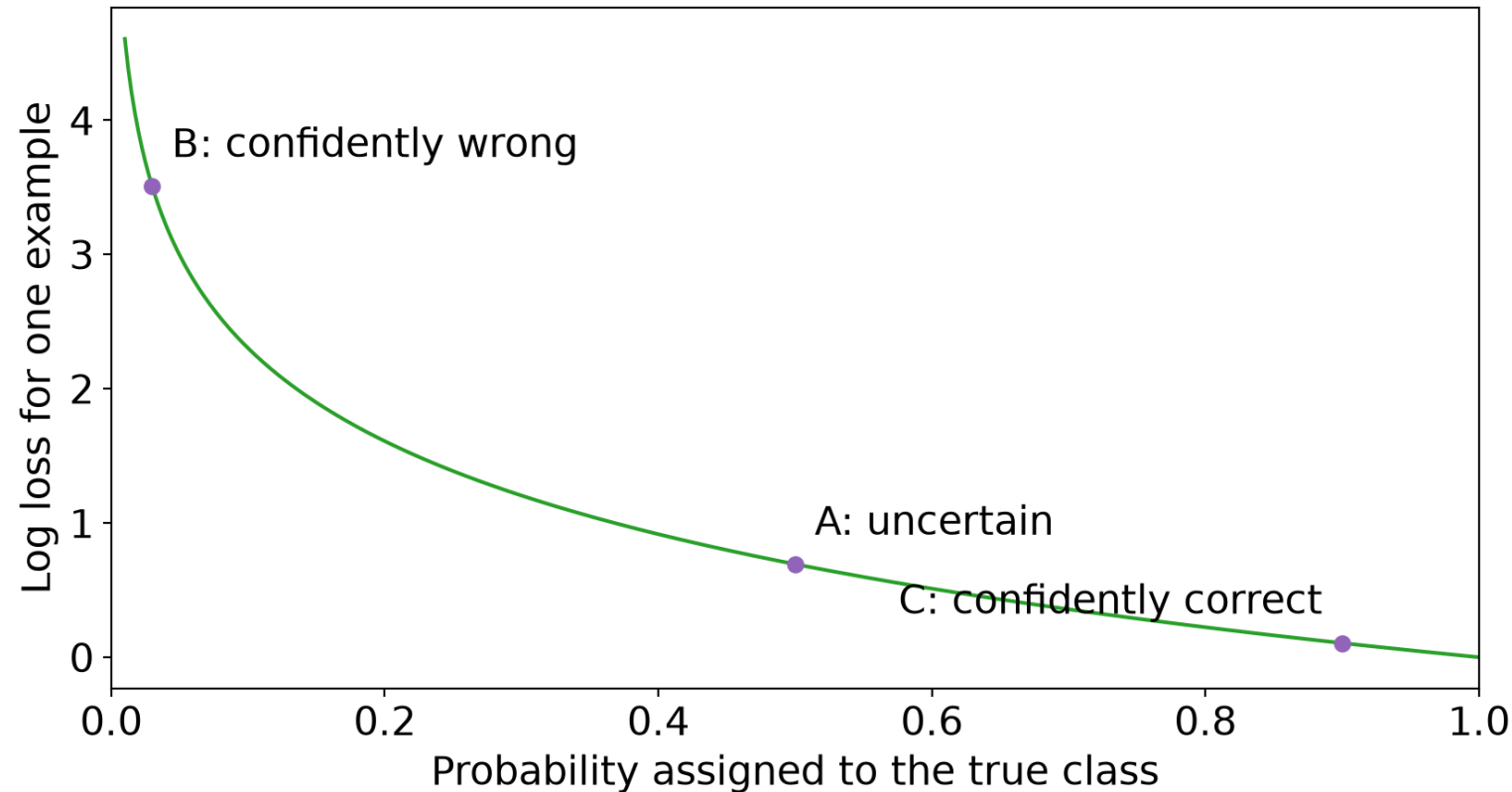
Log loss for our cat example

The true class is **cat**, so $L = -\log(P(\text{cat}))$.

Model	$P(\text{cat})$	Log loss	Interpretation
A	0.50	0.69	Uncertain
B	0.03	3.51	Confidently wrong
C	0.90	0.11	Confidently correct

- Assigning probability 1 to the true class gives loss 0.
- As the probability assigned to the true class approaches 0, the loss grows without bound.

Log loss penalizes confident mistakes



Increasing the probability assigned to the true class reduces the loss.

(Optional) Binary log loss

Encode **cat** as $y = 1$ and **dog** as $y = 0$. Let $p = P(y = 1 \mid \mathbf{x})$.

$$L = -[y \log(p) + (1 - y) \log(1 - p)]$$

- True class is cat ($y = 1$): $L = -\log(p)$.
- True class is dog ($y = 0$): $L = -\log(1 - p)$.
- In both cases, this is **negative log probability of the true class**.

Dataset log loss is the **average** loss across examples. Logistic regression minimizes training log loss plus a regularization penalty.

(Optional) Extending log loss to multiclass classification

For K classes, let $\hat{p}_k$ be the predicted probability of class k . The one-hot target has $y_k = 1$ for the true class and 0 otherwise.

$$L = - \sum_{k=1}^K y_k \log(\hat{p}_k) = -\log(\hat{p}_{\text{true}})$$

For class order **cat, dog, wild**:

- True class cat: $\mathbf{y} = [1, 0, 0]$.
- Predicted probabilities: $\hat{\mathbf{p}} = [0.9, 0.07, 0.03]$.
- Only the cat term contributes: $L = -\log(0.9) \approx 0.11$.

The same idea applies: **reward probability assigned to the true class**.

Your turn: Is a log loss of 0.69 good?

A binary classifier has a mean training log loss of 0.69. Has it learned much?

Regularization: α vs. C

Model	Stronger regularization	Weaker regularization
Ridge	Increase α	Decrease α
LogisticRegression	Decrease C	Increase C

- C controls **inverse** regularization strength.
- Too much regularization can cause underfitting; more is not always better.
- Select hyperparameters using cross-validation.
- Probabilities and log loss can change even when class predictions and accuracy stay the same.

Prediction probabilities and model limitations

- Near the decision boundary: scores near zero, probabilities near 0.5.
- Farther from the boundary: more extreme probabilities.
- A model can be **confidently wrong** if its linear boundary cannot represent the actual relationship.
- Example: a straight Canada–USA boundary can confidently misclassify cities in Alaska.
- A high `predict_proba` value reflects the model's estimate, not a guarantee.

Training loss vs. evaluation score

Model	Fitting objective	Default <code>.score()</code>
<code>Ridge</code>	Squared-error loss + penalty	R^2
<code>LogisticRegression</code>	Log loss + penalty	Accuracy

- `fit` uses its objective to learn coefficients from the training data.
- `score` evaluates predictions; it need not report the fitting loss.
- By default, `cross_validate` uses the estimator's `.score()`.
- Low training loss does **not** guarantee good predictions on new examples.

Your turn: How many parameters?

- Imagine you are training a binary logistic regression model. For each of the following scenarios, identify how many parameters (weights and biases) will be learned.
- Scenario 1: 100 features and 1,000 examples
- Scenario 2: 100 features and 1 million examples

Parametric

- Examples: Logistic regression, linear regression, linear SVM
- Models with a fixed number of parameters, regardless of the dataset size
- Fast predictions; useful for large, sparse feature matrices
- Less flexible, may not capture complex relationships

Non parametric

- Examples: KNN, SVM RBF, Decision tree with no specific depth specified
- The stored representation can grow with the training set; k -NN retains training examples.
- Flexible, can adapt to complex patterns
- Prediction can be expensive; local structure and distances must be useful.

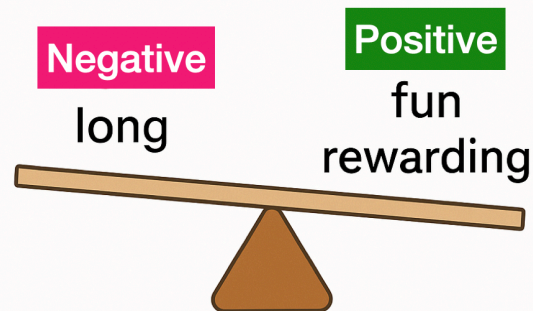
Parametric does not mean assumption-free or unable to overfit

- With 100 features, binary logistic regression learns **101 parameters** in both scenarios.
- With 100,000 features, it learns 100,001 parameters and may overfit.
- Non-parametric models still have assumptions and hyperparameters, such as k and the distance measure in k -NN.
- A linear model assumes its represented features support a useful linear rule; it does not require normally distributed features or targets.

Sentiment analysis example

Class demo

The lectures are long but the demos are fun and the assignments are rewarding.



$$\hat{y} = \sigma(1 \times 0.61 + 1 \times 0.54 + \dots + 1 \times (-0.68))$$

	Coefficient
excellent	0.79
amazing	0.62
fun	0.61
rewarding	0.54
...	...
long	-0.68
boring	-0.73
awful	-0.87
worst	-1.13

- Logistic regression learns **coefficients for each word** from training data.
- Positive coefficients → push prediction toward positive class.
- Negative coefficients → push prediction toward negative class.
- In this example, positive words (*fun*, *rewarding*) outweigh the negative word (*long*), so the overall sentiment is likely **positive**.

(iClicker) Exercise 7.1

Select all of the following statements which are TRUE.

- a. Increasing the hyperparameter `alpha` of `Ridge` is likely to decrease model complexity.
- b. `Ridge` can be used with datasets that have multiple features.
- c. With `Ridge`, we learn one coefficient per training example.
- d. If you train a linear regression model on a 2-dimensional problem (2 features), the model will learn 3 parameters: one for each feature and one for the bias term.

(iClicker) Exercise 7.2

Select all of the following statements which are TRUE.

- a. Increasing logistic regression's C hyperparameter increases model complexity.
- b. The raw output score can be used to calculate the probability score for a given prediction.
- c. For linear classifier trained on d features, the decision boundary is a $d - 1$ -dimensional hyperplane.
- d. A linear model is likely to be uncertain about the data points close to the decision boundary.

Summary

- Linear models use a weighted sum of features and an intercept.
- Loss tells `fit` what counts as a better prediction.
- Squared-error loss emphasizes large residuals; log loss emphasizes confident mistakes.
- Regularization balances fit against coefficient size: larger `alpha` or smaller `C` means a stronger penalty.
- Training loss and evaluation score serve different purposes; use validation to assess generalization.
- Coefficients describe the model, and high predicted probabilities do not guarantee correctness.

Coming up

A few big questions remain:

- How do we tune hyperparameters?
- How do we choose our features? (feature selection, dimensionality reduction, regularization etc.)
- How do we come up with new useful features (feature engineering)
- How do we choose between different models? (different evaluation metrics, what happens if we're not happy with our test error?)
- How to deal with class imbalance?
- What do we do if we do not have targets (unsupervised learning)
- How to build models for more interesting data such as images, user preferences, or sequential data?)

Exit activity (time-permitting)

So far, we have worked with various transformers and supervised machine learning models. The goal of this activity is collaboratively complete tables that provide an overview of

1. the strengths, weaknesses, key hyperparameters of different machine learning models
2. the purpose, use cases, and key considerations of various transformers

(This will serve as a handy reference for your upcoming exam and beyond!)

Activity description

For strengths and weaknesses, some things to consider are:

- concerns about underfitting
- concerns about overfitting
- speed
- scalability for large data sets
- interpretability
- effectiveness on sparse data
- ease of use for multi-class classification
- ability to represent uncertainty
- time/space complexity
- etc.

Predictive estimators

Fill in the following table with at least one entry per box.

Model	Strengths	Weaknesses	Key hyperparameters
decision tree			
k -NN			
RBF SVM			
linear models			

Transformers

Transformation	Purpose	Use cases	Key consideration
Imputation			
Scaling			
One-hot encoding			
Ordinal encoding			
Bag-of-words encoding			

(Optional) Multinomial probabilities with softmax

Softmax Function for Probabilities

Given an input, the probability that it belongs to class $j \in \{1, 2, \dots, K\}$ is calculated using the **softmax function**:

$$P(y = j \mid x_i) = \frac{e^{w_j^\top x_i + b_j}}{\sum_{k=1}^K e^{w_k^\top x_i + b_k}}$$

- x_i is the i^{th} example
- w_j is the weight vector for class j .
- b_j is the bias term for class j .
- K is the total number of classes.

Making Predictions

1. Compute Probabilities:

For each class j , compute the probability $P(y = j \mid x_i)$ using the softmax function.

2. Select the Class with the Highest Probability:

The predicted class $\hat{y}$ is:

$$\hat{y} = \arg \max_{j \in \{1, \dots, K\}} P(y = j \mid x_i)$$

Binary vs multinomial logistic regression

Aspect	Binary Logistic Regression	Multinomial Logistic Regression
Target variable	2 classes (binary)	More than 2 classes (multi-class)
Getting probabilities	Sigmoid	Softmax
parameters	d weights, one per feature and the bias term	d weights and a bias term per class
Output	Single probability	Probability distribution over classes
Use case	Binary classification (e.g., spam detection)	Multi-class classification (e.g., flower species)