

# CPSC 330 Lecture 8: Hyperparameter Optimization

Firas Moosvi (Slides adapted from Varada Kolhatkar)



# Recap: Logistic regression

- A linear model used for binary classification tasks.
  - There is a variant of logistic regression called multinomial logistic regression for multiclass classification.
- Parameters:
  - Coefficients (Weights): The model learns a coefficient or a weight associated with each feature that represents its importance.
  - Bias (Intercept): A constant term added to the linear combination of features and their coefficients.

# Recap: Logistic regression

- The model computes a weighted sum of the input features' values, adjusted by their respective coefficients and the bias term.
- This weighted sum is passed through a sigmoid function to transform it into a probability score, indicating the likelihood of the input belonging to the “positive” class.

$$P_{hat} = \sigma \left( \sum_{i=1}^d w_i x_i + b \right)$$

- $P_{hat}$  is the predicted probability of the example belonging to the positive class.
- $w_i$  is the learned weight associated with feature  $i$
- $x_i$  is the value of the input feature  $i$
- $b$  is the bias term

# Recap: Logistic regression

- For a dataset with  $d$  features, the decision boundary that separates the classes is a  $d - 1$  dimensional hyperplane.
- Complexity hyperparameter:  $C$  in `sklearn`.
  - Higher  $C \rightarrow$  more complex model meaning larger coefficients
  - Lower  $C \rightarrow$  less complex model meaning smaller coefficients

# Recap: `CountVectorizer` input

- Primarily designed to accept either a `pandas.Series` of text data or a 1D `numpy` array. It can also process a list of string data directly.
- Unlike many transformers that handle multiple features (`DataFrame` or 2D `numpy` array), `CountVectorizer` a single text column at a time.
- If your dataset contains multiple text columns, you will need to instantiate separate `CountVectorizer` objects for each text feature.
- This approach ensures that the unique vocabulary and tokenization processes are correctly applied to each specific text column without interference.

# Hyperparameter optimization motivation

# Data

```

1 sms_df = pd.read_csv(DATA_DIR + "spam.csv", encoding="latin-1")
2 sms_df = sms_df.drop(columns = ["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"])
3 sms_df = sms_df.rename(columns={"v1": "target", "v2": "sms"})
4 train_df, test_df = train_test_split(sms_df, test_size=0.10, random_state=42)
5 X_train, y_train = train_df["sms"], train_df["target"]
6 X_test, y_test = test_df["sms"], test_df["target"]
7 train_df.head(4)

```

	target	sms
3130	spam	LookAtMe!: Thanks for your purchase of a video...
106	ham	Aight, I'll hit you up when I get some cash
4697	ham	Don no da:)whats you plan?
856	ham	Going to take your babe out ?



# Model building

- Let's define a pipeline

```
1 pipe_svm = make_pipeline(CountVectorizer(), SVC())
```

- Suppose we want to try out different hyperparameter values.

```
1 parameters = {  
2     "max_features": [100, 200, 400],  
3     "gamma": [0.01, 0.1, 1.0],  
4     "C": [0.01, 0.1, 1.0],  
5 }
```

# Hyperparameter optimization with loops

- Define a parameter space.
- Iterate through possible combinations.
- Evaluate model performance.
- What are some limitations of this approach?

# sklearn methods

- `sklearn` provides two main methods for hyperparameter optimization
  - Grid Search
  - Random Search

# Grid Search

- Covers all possible combinations from the provided grid.
- Can be parallelized easily.
- Integrates cross-validation.

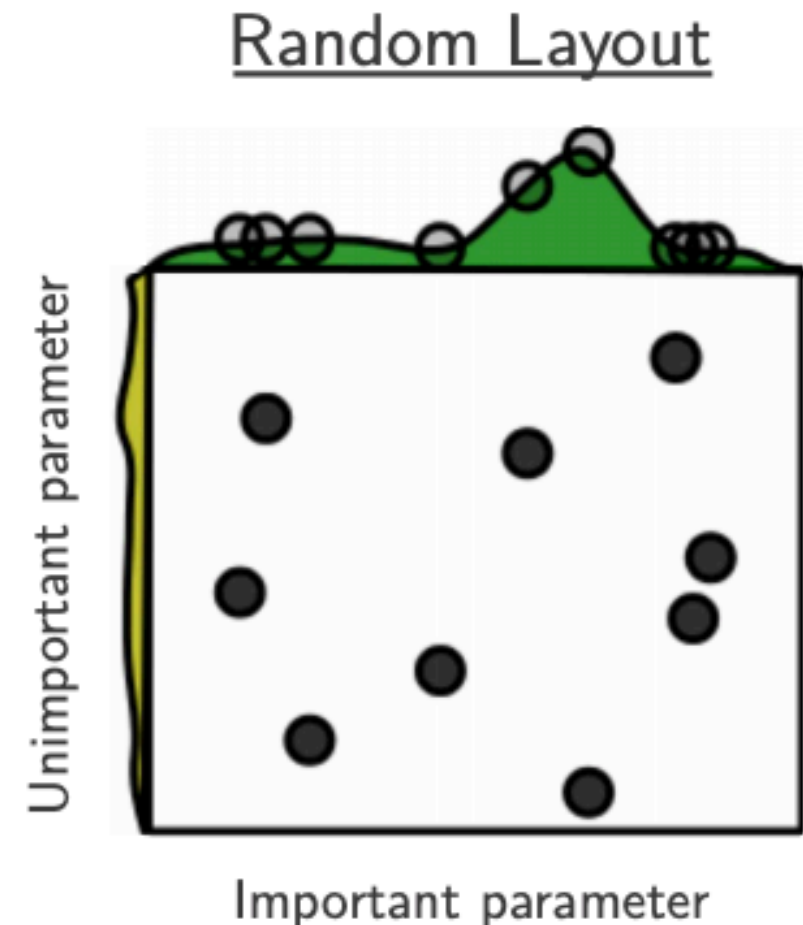
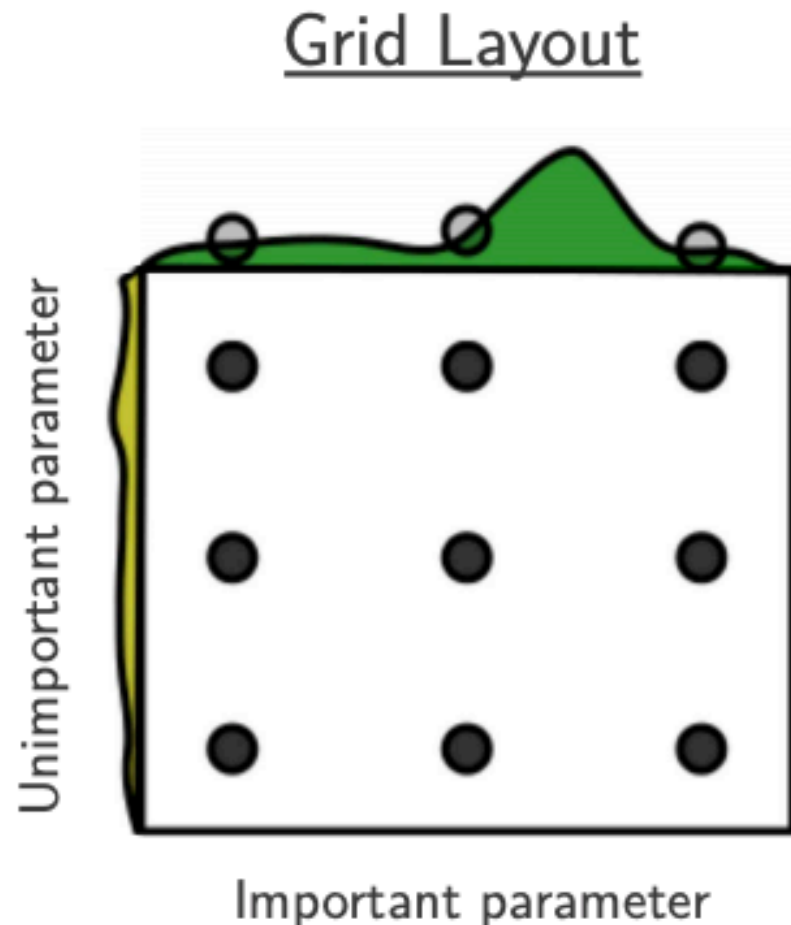
# Grid search example

```
1 from sklearn.model_selection import GridSearchCV
2
3 pipe_svm = make_pipeline(CountVectorizer(), SVC())
4
5 param_grid = {
6     "countvectorizer__max_features": [100, 200, 400],
7     "svc__gamma": [0.01, 0.1, 1.0],
8     "svc__C": [0.01, 0.1, 1.0],
9 }
10 grid_search = GridSearchCV(pipe_svm,
11                             param_grid = param_grid,
12                             n_jobs=-1,
13                             return_train_score=True
14                             )
15 grid_search.fit(X_train, y_train)
16 grid_search.best_score_
```

```
np.float64(0.9780612255051213)
```

# Random Search

- More efficient than grid search when dealing with large hyperparameter spaces.
- Samples a given number of parameter settings from distributions.



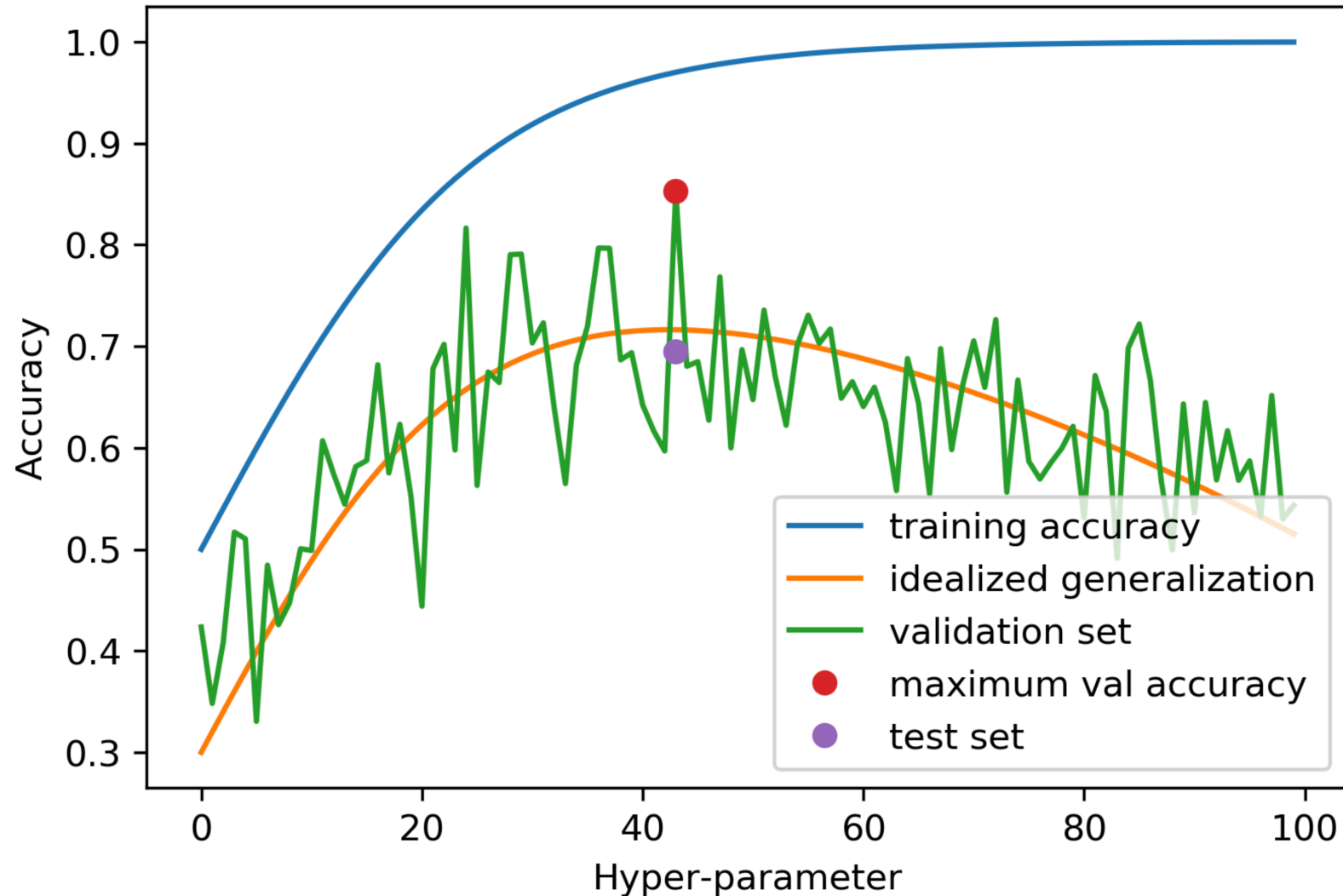
# Random search example

```
1 from sklearn.model_selection import RandomizedSearchCV
2 from scipy.stats import randint
3
4 pipe_svc = make_pipeline(CountVectorizer(), SVC())
5
6 param_dist = {
7     "countvectorizer__max_features": randint(100, 2000),
8     "svc__C": uniform(0.1, 1e4), # loguniform(1e-3, 1e3),
9     "svc__gamma": loguniform(1e-5, 1e3),
10 }
11 random_search = RandomizedSearchCV(pipe_svc,
12     param_distributions = param_dist,
13     n_iter=10,
14     n_jobs=-1,
15     return_train_score=True)
16
17 # Carry out the search
18 random_search.fit(X_train, y_train)
19 random_search.best_score_
```

np.float64(0.9834462679824798)

# Optimization bias

- Why do we need separate validation and test datasets?





# Mitigating optimization bias.

- Cross-validation
- Ensembles
- Regularization and choosing a simpler model

# (iClicker) Exercise 8.1

iClicker cloud join link: <https://join.iclicker.com/YJHS>

Select all of the following statements which are TRUE.

- a. If you get best results at the edges of your parameter grid, it might be a good idea to adjust the range of values in your parameter grid.
- b. Grid search is guaranteed to find the best hyperparameter values.
- c. It is possible to get different hyperparameters in different runs of RandomizedSearchCV.

# Questions for you

- You have a dataset and you give me 1/10th of it. The dataset given to me is rather small and so I split it into 96% train and 4% validation split. I carry out hyperparameter optimization using a single 4% validation split and report validation accuracy of 0.97. Would it classify the rest of the data with similar accuracy?
  - Probably
  - Probably not

# Questions for class discussion

- Suppose you have 10 hyperparameters, each with 4 possible values. If you run `GridSearchCV` with this parameter grid, how many experiments will be carried out?
- Suppose you have 10 hyperparameters and each takes 4 values. If you run `RandomizedSearchCV` with this parameter grid with `n_iter=20`, how many cross-validation experiments will be carried out?

# Group Work: Class Demo & Live Coding

For this demo, each student should [click this link](#) to create a new repo in their accounts, then clone that repo locally to follow along with the demo from today.

# Looking Ahead

# Group Work: Invention Activity

So far we have looked only at **score** as a metric for evaluating our metrics.

What else could be used as a possible metric? Think of what else might be important for machine learning practitioners and stakeholders?

In your group, brainstorm 4 alternative options:

- 
- 
- 
-