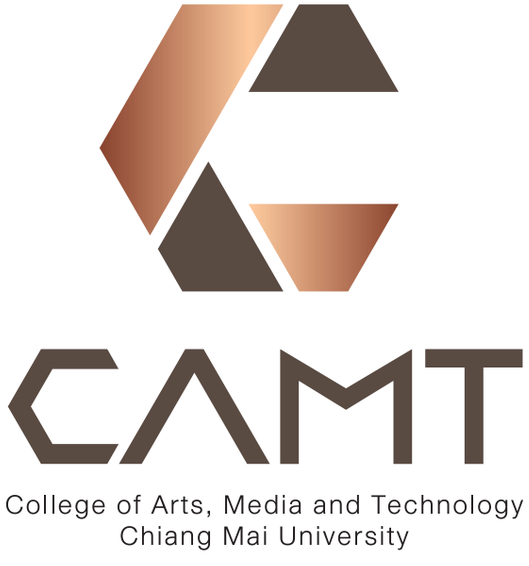
SE 481 Introduction to Information Retrieval (IR for SE)

Module #7 — Machine Learning and IR



Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

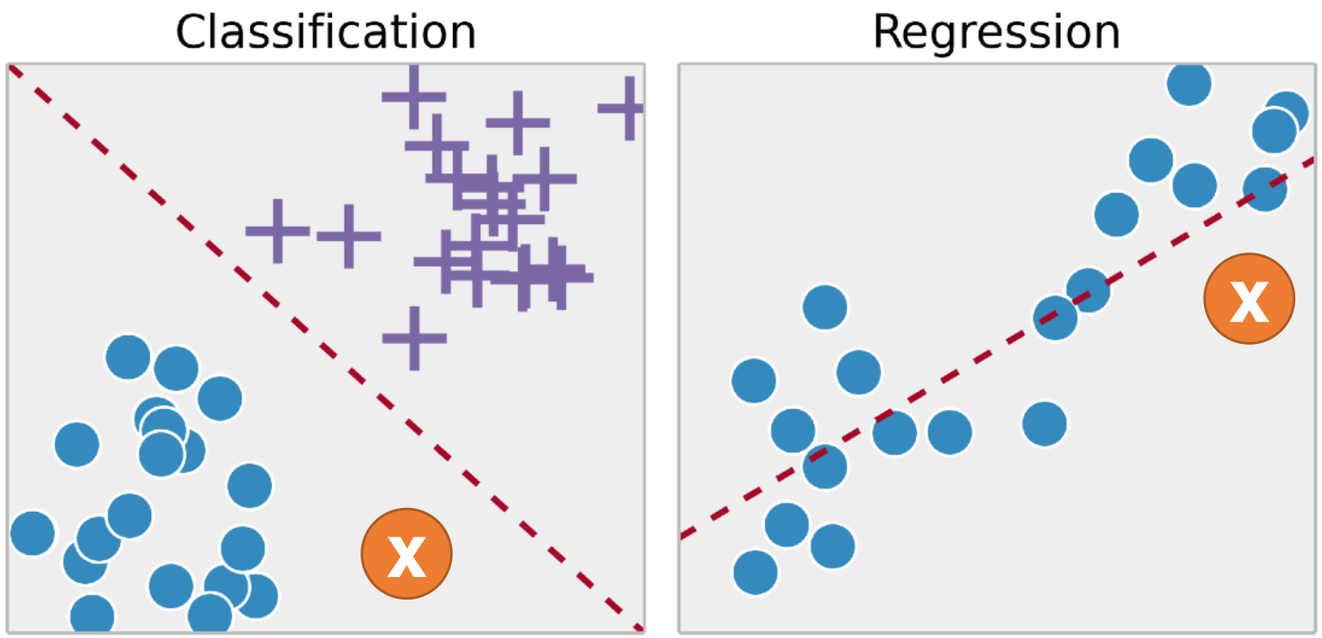
College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Agenda

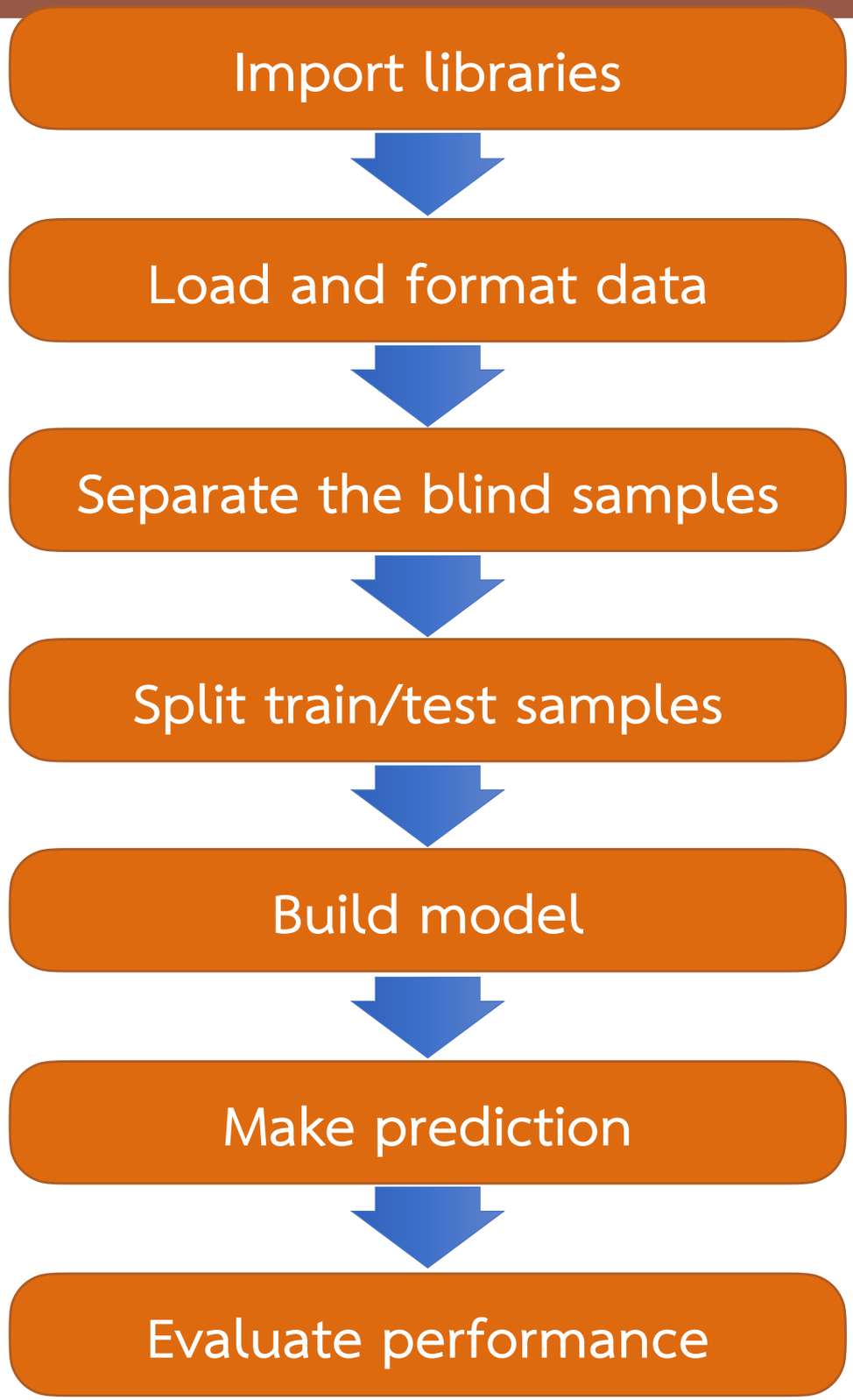
- Brief review in basic machine learning processes
- Exploration of how machine learning is used within various IR tasks.
- The integration of machine learning techniques to improve IR systems.

Regression vs Classification



$x \in \{0,+\}$ $x \in (-\infty,\infty)$

Common model building steps

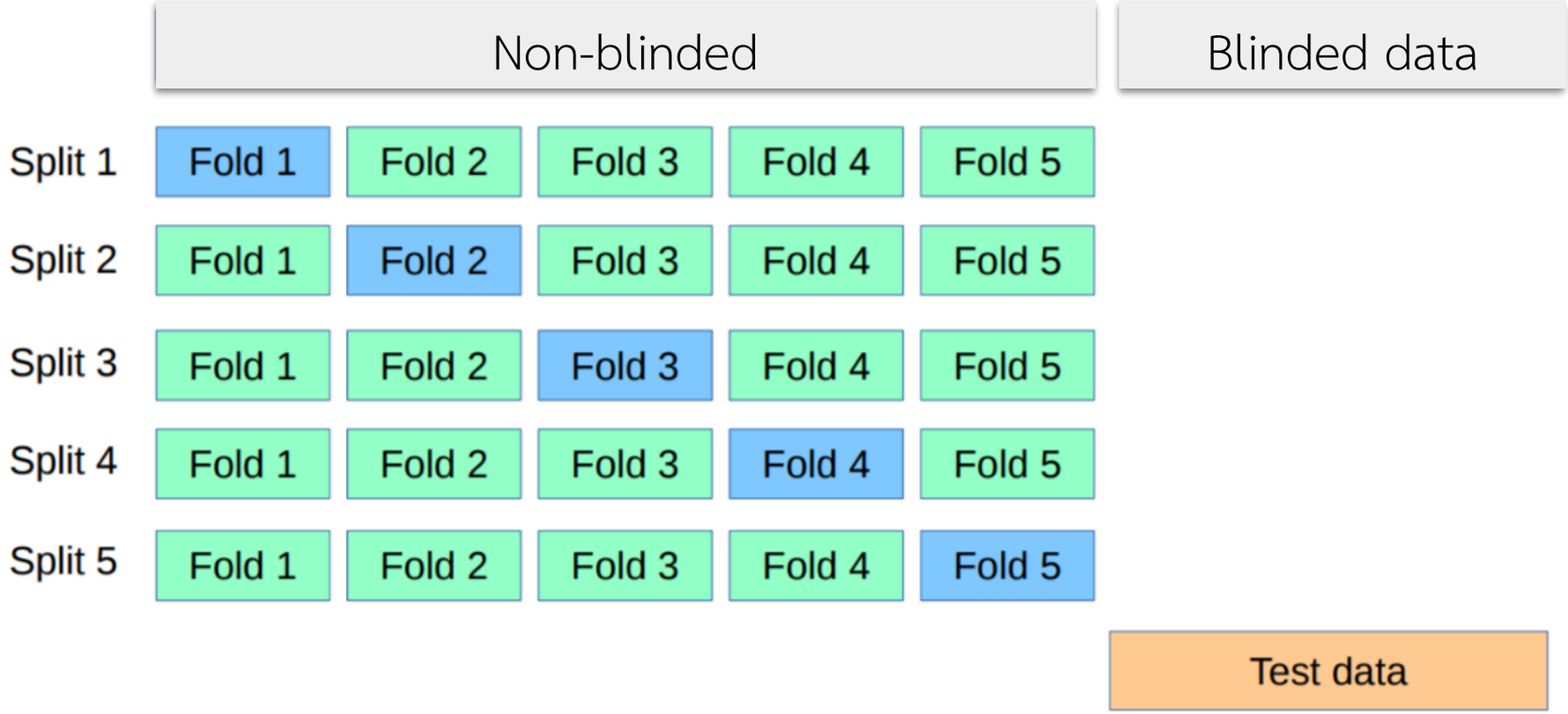


Separate the blind samples

Non-blinded data	Blinded data
eg. 90% of the entries	eg. 10% of the entries

Split train/test samples

- Cross validation, e.g., 5-fold cross validation



Ref: https://scikit-learn.org/stable/modules/cross_validation.html

Common model building steps

- Pipeline

Ref: https://www.slideshare.net/PyData/julie-michelman-pandas-pipelines-and-custom-transformers

In short — Objective of machine learning tasks:

- To minimize error rates — minimizing the differences, i.e., **error** between the actual outcomes and the model's estimations or predictions across all instances evaluated by the trained algorithm.
- Minimizing error is evidenced by the lowest error value obtained through the training phase and its subsequent application in blind testing.

Error(X) = noise(X) + bias(X) + variance(X)

- Error(X) is the sum of noise, bias, and variance in model predictions.
- Bias occurs when a model simplifies the data too much, missing relevant relations and patterns (underfitting).
- Variance arises when a model is too sensitive to the training data, capturing random noise instead of the intended outputs (overfitting).

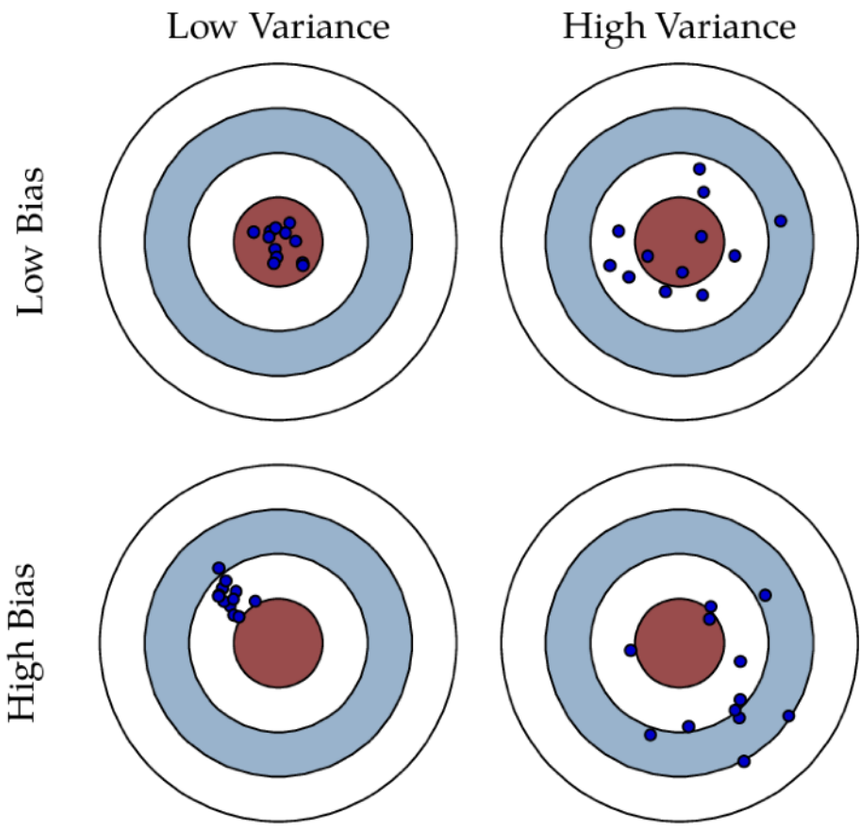
Simplified explanation, e.g.,

- Imagine someone who guesses your question before you finish. That's high bias—they're not listening to everything.
- Similar to someone who comes up with wild, unpredictable responses, making things too complex.
- These traits contribute to a model's unique **personality**, affecting its ability to predict accurately.

Simplified explanation, e.g.,

- **High bias/low variance:** — Comparable to someone who consistently offers **the same incorrect answer**, no matter the question.
- **High bias/high variance** — A person who throws out all sorts of wrong answers.
- **Low bias/high variance** — Resembles a person who tries to get it right and usually do, but sometimes their imagination leads to some wacky ideas.
- **Low bias/low variance** — A thoughtful individual who consistently delivers reliable and accurate answers.

Variance and accuracy



Ref: https://i1.wp.com/www.d4t4v1z.com/wp-content/uploads/2017/09/bias.png?resize=835%2C770

Ensemble

- Utilizes multiple models to offset the weaknesses inherent to a single predictive technique.
- Helps prevent errors that happen when one model doesn't work well with certain data.
- Generally classified into
  - Uses one model type across different data segments, like Random Forest.
  - Integrates diverse types of models, as seen in stacking or stack generalization.

In theory

- The Mean Squared Error (MSE) of one single model is due to bias and variance
- Theoretical models demonstrate that averaging predictions across models can reduce the overall MSE

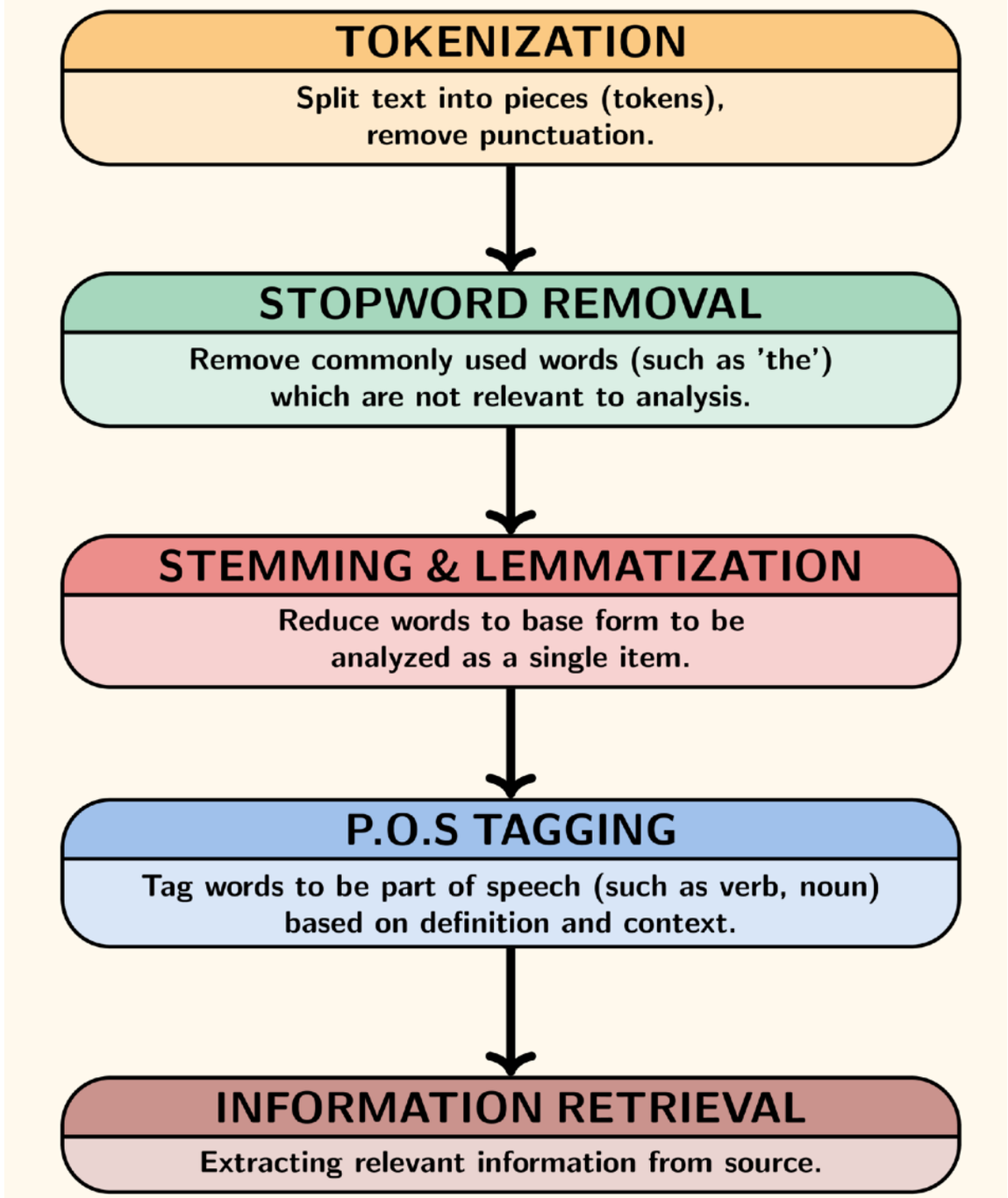
$$MSE = \overline{Bias^2} + \frac{1}{m}\overline{Var} + \left(1 - \frac{1}{m}\right)\overline{Covar}$$

- The concern over an individual model's variance diminishes if the ensemble consists of independent and unbiased models.

Ref: G.Brown, J.L.Wyatt, and P.Tino, "Managing diversity in regression ensembles" J. Machine Learning Research, vol. 6, no. Sep, pp. 1621–1650, 2005.

Ref: X. Zhu, P. Zhang, X. Lin, and Y. Shi, "Active learning from stream data using optimal weight classifier ensemble" IEEE Trans. Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 40, no. 6, pp. 1607–1621, 2010.

How machine learning is used within various IR tasks



Case study — Bug report notation

- https://www.kaggle.com/anmolkumar/github-bugs-prediction
- A Kaggle dataset.
- Participants were tasked with developing an algorithm that **classifies GitHub issues** into **bugs**, **features**, and **questions**, using **issue titles** and **descriptions**.

Dataset description

- **embold_train.json** — Contains 150,000 records across 3 columns, including the target variable for labels.
- **embold_test.json** — Comprises 30,000 instances with 2 columns, used for model validation.
- **embold_train_extra.json** — An expanded set with 300,000 entries, also featuring 3 columns with labels.
- Sample **sample submission.csv** — Refer to the Evaluation section for guidance on creating a proper submission.

Attribute description

- **Title** — The heading provided for each GitHub issue, indicating the nature of the content.
- **Body** — More info on what's going on or what's being suggested in the post.
- **Label** — Categorical variable representing the classification of the issue:
  - Bug - 0
  - Feature - 1
  - Question - 2

To do

- Construct a predictive model capable of determining if a GitHub message pertains to **a bug** or is related to **other categories.**

Some process walkthroughs

```
01  def preprocess(text):
02      cleaned_text = text.translate(str.maketrans('', '', '!"#$%&\'()*+,.<=>?@[]^`{|}~' + u'\xa0'))
03      cleaned_text = cleaned_text.lower()
04      cleaned_text = cleaned_text.translate(str.maketrans(string.whitespace, ' ' * len(string.whitespace), ''))
05
06      cleaned_text = ' '.join(['_variable_with_underscore' if '_' in t else t for t in cleaned_text.split()])
07      cleaned_text = ' '.join(['_variable_with_dash' if '-' in t else t for t in cleaned_text.split()])
08      cleaned_text = ' '.join(['_long_variable_name' if len(t) > 15 and t[0] != '#' else t for t in
cleaned_text.split()])
09      cleaned_text = ' '.join(['_weburl' if t.startswith('http') and '/' in t else t for t in cleaned_text.split()])
10      cleaned_text = ' '.join(['_number' if re.sub('[\\/;:_-]', '', t).isdigit() else t for t in cleaned_text.split()])
11      cleaned_text = ' '.join(['_variable_with_address' if re.match('.*0x[0-9a-f].*', t) else t for t in
cleaned_text.split()])
12      cleaned_text = ' '.join(['_name_with_number' if re.match('.*[a-f]*:[0-9]*', t) else t for t in
cleaned_text.split()])
13      cleaned_text = ' '.join(['_number_starts_with_one_character' if re.match('[a-f][0-9].*', t) else t for t in
cleaned_text.split()])
14      cleaned_text = ' '.join(['_number_starts_with_three_characters' if re.match('[a-f]{3}[0-9].*', t) else t for t in
cleaned_text.split()])
15      cleaned_text = ' '.join(['_version' if any(i.isdigit() for i in t) and t.startswith('v') else t for t in
cleaned_text.split()])
16      cleaned_text = ' '.join(['_localpath' if ('\\' in t or '/' in t) and ':' not in t else t for t in
cleaned_text.split()])
17      cleaned_text = ' '.join(['_image_size' if t.endswith('px') else t for t in cleaned_text.split()])
18
19      tokenized_text = word_tokenize(cleaned_text)
20      sw_removed_text = [word for word in tokenized_text if word not in stopword_set]
21      sw_removed_text = [word for word in sw_removed_text if len(word) > 2]
22      stemmed_text = ' '.join([stemmer.stem(w) for w in sw_removed_text])
23
24      return stemmed_text
```


Similar to that of the first and the second modules of this course

Some process walkthroughs

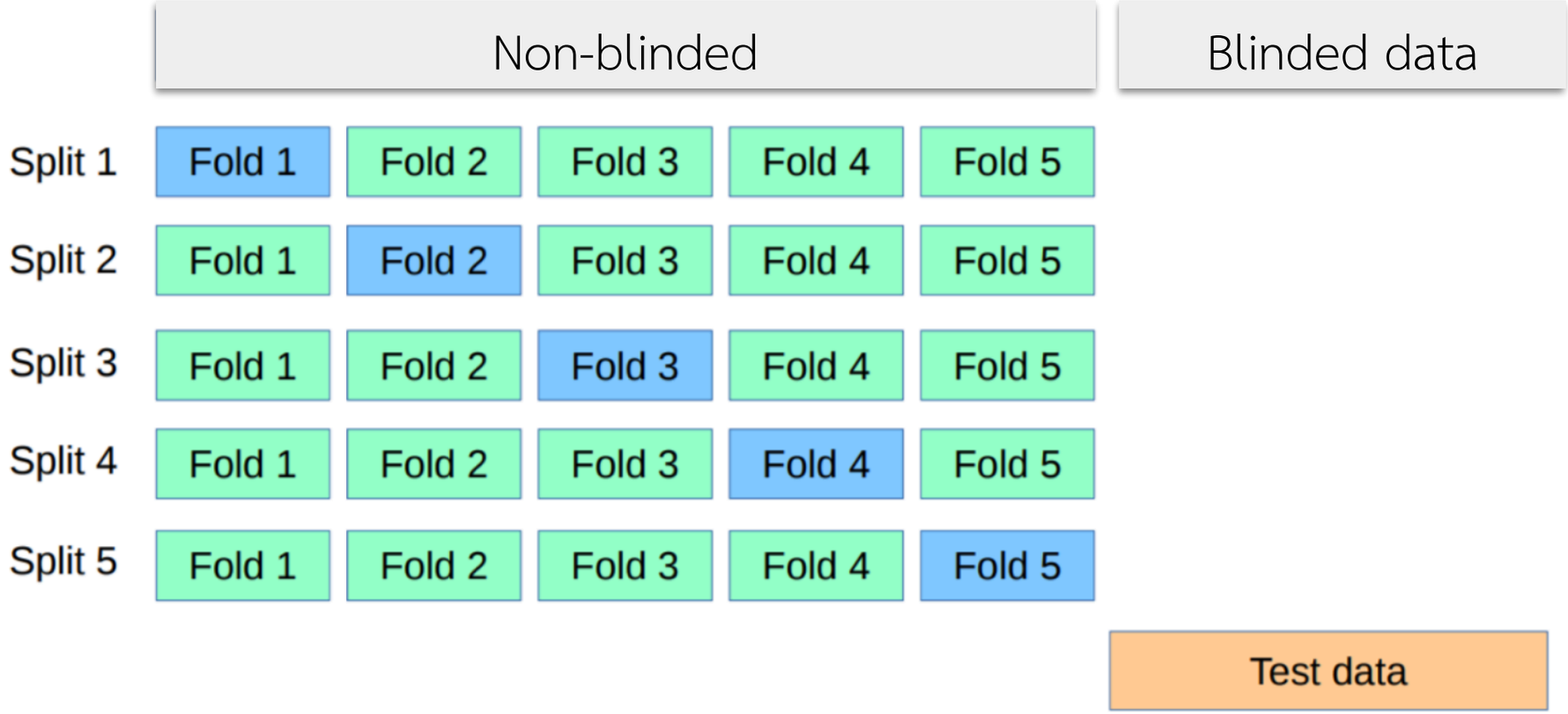
```
from multiprocessing.pool import ThreadPool as Pool


1   dataset = pd.read_json('resource/embold_train.json')
2   dataset.loc[dataset['label'] > 0, 'label'] = -1
3   dataset.loc[dataset['label'] == 0, 'label'] = 1
4   dataset.loc[dataset['label'] == -1, 'label'] = 0
5   stopwords = set(stopwords.words('English'))
6   ps = PorterStemmer()
7   pool = Pool(8, initializer=initialize_pool, initargs=(stopwords, ps, ))
8
9   cleaned_title = pool.map(preprocess, dataset.title)
10  cleaned_body = pool.map(preprocess, dataset.body)
```

Parallel the IR preprocess

Common model building steps

- Cross validation, e.g., 5-fold cross validation



Ref: https://scikit-learn.org/stable/modules/cross_validation.html

Some process walkthroughs

```
01  data_texts = pd.DataFrame([cleaned_title, cleaned_body], index=['title','body']).T
02  y = dataset['label']
03
04  data_fit, data_blindtest, y_fit, y_blindtest = model_selection.train_test_split(data_texts, y,
test_size=0.1)
05
06  tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,1))
07  tfidf_vectorizer.fit(cleaned_title + cleaned_body)
08
09  X_tfidf_fit = tfidf_vectorizer.transform(data_fit['title])
10  X_tfidf_blindtest = tfidf_vectorizer.transform(data_blindtest['title'])
```

◀····· Try bigram if you have an access to high computational power.

Transform the preprocess texts into ngrams of tf-idf

Some process walkthroughs

To use the #number of total CPU cores minus 2

```
01  gbm_model = lgb.LGBMClassifier()
02
03  precision_cv_score = model_selection.cross_val_score(gbm_model, X_tfidf_fit, y_fit, cv=5,
    n_jobs=-2, scoring='precision_macro').mean()
04  recall_cv_score = model_selection.cross_val_score(gbm_model, X_tfidf_fit, y_fit, cv=5, n_jobs=-2,
    scoring='recall_macro').mean()
05  f1_cv_score = model_selection.cross_val_score(gbm_model, X_tfidf_fit, y_fit, cv=5, n_jobs=-2,
    scoring='f1_macro').mean()
06
07  print('CV: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_cv_score, recall_cv_score, f1_cv_score))
```

Try CV>5 if you have an access to a high computing power.

Some process walkthroughs

```
01  data_fit_train, data_fit_test, y_fit_train, y_fit_test =
    model_selection.train_test_split(data_fit, y_fit, test_size=0.3)
02
03  X_tfidf_fit_train = tfidf_vectorizer.transform(data_fit_train['title'])
04  X_tfidf_fit_test = tfidf_vectorizer.transform(data_fit_test['title'])
05  X_tfidf_blindtest = tfidf_vectorizer.transform(data_blindtest['title'])
06
07  gbm_model.fit(X_tfidf_fit_train, y_fit_train, eval_set=[(X_tfidf_fit_test, y_fit_test)],
    eval_metric='AUC')
08
09  precision_test_score = metrics.precision_score(gbm_model.predict(X_tfidf_blindtest), y_blindtest,
    average='macro')
10  recall_test_score = metrics.recall_score(gbm_model.predict(X_tfidf_blindtest), y_blindtest,
    average='macro')
11  f1_test_score = metrics.f1_score(gbm_model.predict(X_tfidf_blindtest), y_blindtest,
    average='macro')
12
13  print('test: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_test_score, recall_test_score,
    f1_test_score))
```

Some process walkthroughs

```
01  pickle.dump(tfidf_vectorizer, open('resource/github_bug_prediction_tfidf_vectorizer.pkl', 'wb'))
02  pickle.dump(gbm_model, open(‘resource/github_bug_prediction_basic_model.pkl', 'wb'))
```


Save the (tf-idf) vectorizer and model

Bug prediction app

```
01  from flask import Flask, request
02  from scipy.sparse import hstack
02  from m7 import preprocess
03  import pickle
04
05  app = Flask(__name__)
06  app.tfidf_vectorizer = pickle.load(open('resource/github_bug_prediction_tfidf_vectorizer.pkl', 'rb')
07  app.basic_model = pickle.load(open('resource/github_bug_prediction_basic_model.pkl', 'rb'))
08  app.stopword_set = set(stopwords.words())
09  app.stemmer = PorterStemmer()
10
11  @app.route('/predict_basic', methods=['GET'])
12  def predict_basic():
13      response_object = {'status': 'success'}
14      argList = request.args.to_dict(flat=False)
15      title = argList['title'][0]
16      body = argList['body'][0]
17      predict =
app.basic_model.predict_proba(hstack([app.tfidf_vectorizer.transform([preprocess(title)])]))
18      response_object['predict_as'] = 'bug' if predict[0][1] > 0.5 else 'not bug'
19      response_object['bug_prob'] = 1 predict[0][1]
20      return response_object
21
22  if __name__ == '__main__':
23      app.run(debug=False)
```

Bug prediction app

```
// 20231113142557
// http://localhost:5000/predict_basic?title=cannot%20download&body=downloads%20failed%20with%20404%20error

{
  "bug_prob": 0.8050498992182438,
  "predict_as": "bug",
  "status": "success"
}
```

	Key	Value
☑	title	cannot download
☑	body	Downloads failed with 404 error
	Key	Value

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize JSON

```
1  {
2    "bug_prob": 0.8050498992182438,
3    "predict_as": "bug",
4    "status": "success"
5  }
```

What's next

- There are still many advanced techniques used in IR and NLP nowadays, e.g.,
  - **Topic Modeling** — For discovering the abstract **topics** that occur in a collection of documents.
  - **Deep Learning** — Leveraging neural networks for complex pattern recognition and prediction

E.g., Topic modeling

- Topic modeling is a statistical method used to identify **abstract themes (main ideas)** within a corpus of text.
- It's a common text-mining technique for uncovering latent semantic patterns in large text collections.
- E.g., in software engineering texts, terms like **commit** and **repository** are commonly associated with **version control**, while **class** and **inheritance** are often discussed in the context of **object-oriented programming**.

E.g., search query: Debugging

- **Content A:** Debugging in software engineering is the process of identifying and removing errors from computer code. The term bug to describe faults in a program dates back to the early days of computing.
- **Content B:** Banished from their integrated development environment (IDE), they embarked on a journey through layers of code to uncover hidden flaws that hindered functionality.
- **Solution**: keyword usage
  - Since Content A contains the word **Debugging** but Content B does not, the engine can easily choose Content A to rank

E.g., search query: Agile methodology

- **Content A:** The **Agile methodology** emphasizes iterative development. It's well-suited for environments that embrace flexible responses to change, like our latest **Agile** sprint.
- **Content B:** If you choose to adopt this framework, you have to speak to the project manager about transitioning to **Agile**.
- **Solution**: *tf-idf*
  - The search engine can use *tf-idf* to detect that **Agile** and **methodology** is a key term in Content A. **Agile** appears frequently where methodology can be more rare in the context. This makes Content A more relevant to the query than Content B.

E.g., search query: Machine learning

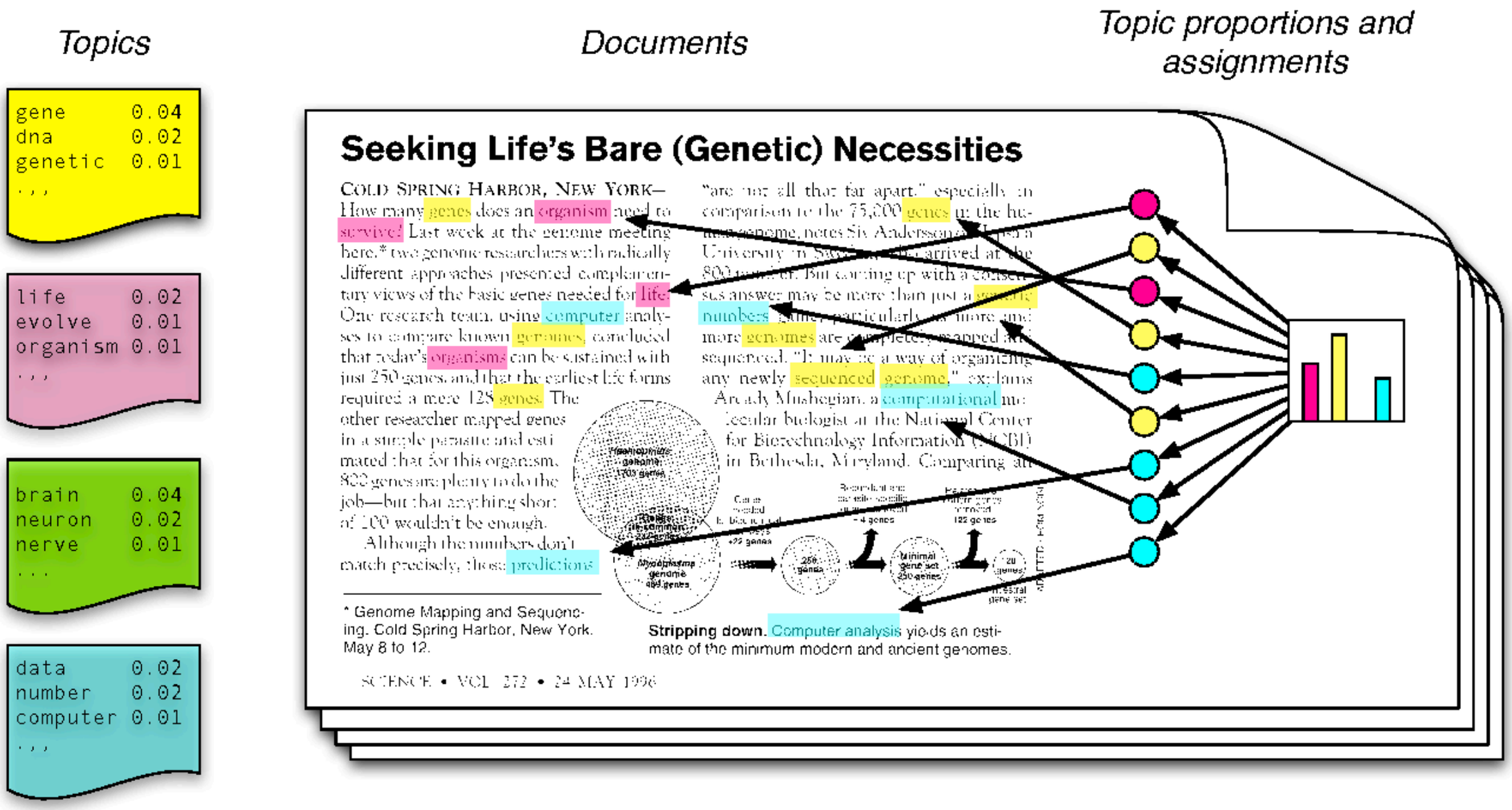
- **Content A:** *The precision of a* ***machine learning*** *model must be refined*, noted the data scientist. Work is ongoing to optimize the algorithm.
- **Content B:** In the realm of artificial intelligence, a neural network, after layers of training, evolves into a **machine learning** powerhouse.
- **Solution**: co-occurrence
  - Using co-occurrence, the engine can determine that phrases like **neural network** and **artificial intelligence** often appear with **machine learning** and thus, Content B is more relevant than Content A.

E.g., search query: Python debugger

- **Content A:** He combed through the log files, searching for the error that caused the system crash, but the root cause remained elusive.
- **Content B:** With each line of code, the developer traced the execution flow, the debugger highlighting the faulty function causing the exception

- **Solution**: Topic modelling
  - As humans reading both sentences, we can infer that Content B is about **troubleshooting software** - debugging in python - and the process involved in it.
  - But a search engine, using only basic keyword matching, might struggle since both sentences use terms like error, system, and execution, which are common in software development discussions.
  - Topic modeling helps discern that **Content B** is more relevant for someone looking for information on a Python debugger.

E.g., Topic modeling



Ref: Blei, D.M. (2010). Introduction to Probabilistic Topic Models.

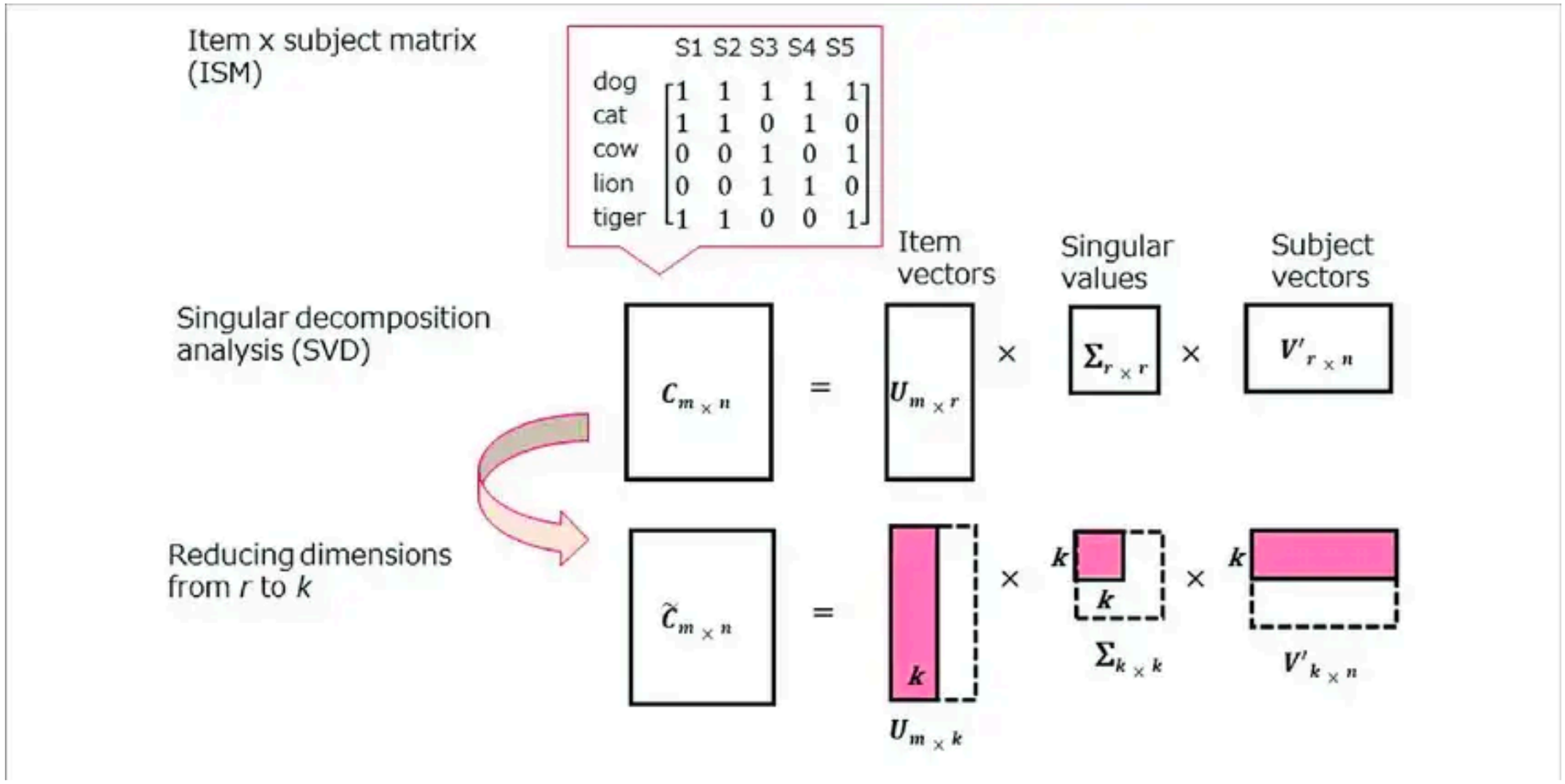
Breaking down Topic modeling

- Uses methods like Latent semantic analysis (LSA) to simplify complex text data by identifying underlying patterns.
- It can also use algorithms such as Latent dirichlet allocation (LDA) that consider the probability of topics within documents, effectively uncovering the distribution of topics across a corpus.

Latent semantic analysis (LSA)

- Implements Truncated Singular value decomposition (SVD) to streamline large text data.
- This technique assumes that the reduced set of features captures the essence of the complete *tf-idf* matrix.
- SVD focuses on extracting the most significant information, representing the original data in a lower-dimensional space without substantial loss of meaning.

Latent semantic analysis (LSA)



Ref: Sumiyoshi, C., Fujino, H., Sumiyoshi, T., Yasuda, Y., Yamamori, H., Fujimoto, M., & Hashimoto, R. (2018). Semantic memory organization in Japanese patients with schizophrenia examined with category fluency. Frontiers in psychiatry, 9, 87.

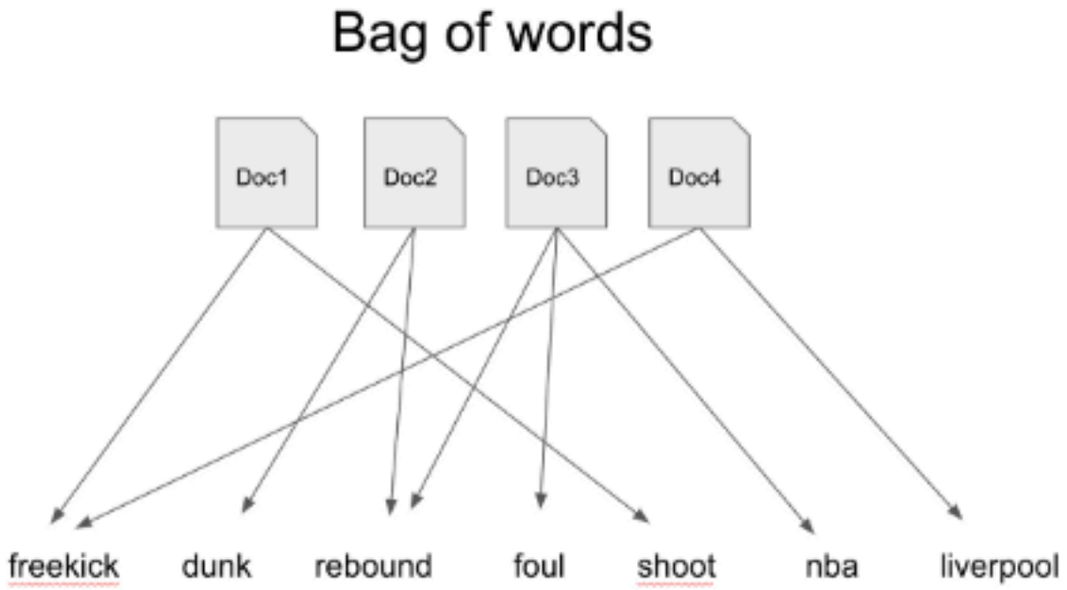
Latent semantic analysis (LSA)

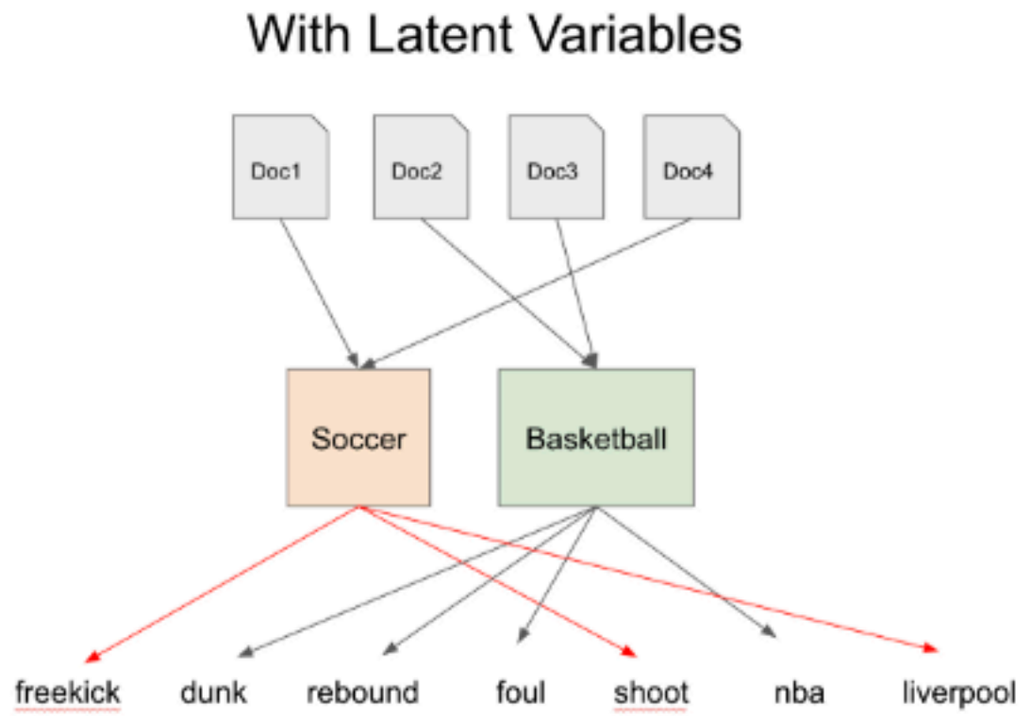
```
01  from sklearn.decomposition import TruncatedSVD
02
03  lsa = TruncatedSVD(n_components=500, n_iter=100, random_state=0)
04  lsa.fit(X_tfidf_fit)
05  X_lsa_fit = lsa.transform(X_tfidf_fit)
06
07  gbm_model_with_lsa = lgb.LGBMClassifier()
08
09  precision_cv_score = model_selection.cross_val_score(gbm_model_with_lsa, X_lsa_fit, y_fit, cv=5,
    n_jobs=-2, scoring='precision_macro').mean()
10  recall_cv_score = model_selection.cross_val_score(gbm_model_with_lsa, X_lsa_fit, y_fit, cv=5,
    n_jobs=-2, scoring='recall_macro').mean()
11  f1_cv_score = model_selection.cross_val_score(gbm_model_with_lsa, X_lsa_fit, y_fit, cv=5, n_jobs=-2,
    scoring='f1_macro').mean()
12
13  print('fit: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_cv_score, recall_cv_score, f1_cv_score))
14
15  X_fit_with_lsa = hstack([X_tfidf_fit, X_lsa_fit]).tocsr()
16
17  precision_cv_score = model_selection.cross_val_score(gbm_model_with_lsa, X_fit_with_lsa, y_fit, cv=5,
    n_jobs=-2, scoring='precision_macro').mean()
18  recall_cv_score = model_selection.cross_val_score(gbm_model_with_lsa, X_fit_with_lsa, y_fit, cv=5,
    n_jobs=-2, scoring='recall_macro').mean()
19  f1_cv_score = model_selection.cross_val_score(gbm_model_with_lsa, X_fit_with_lsa, y_fit, cv=5,
    n_jobs=-2, scoring='f1_macro').mean()
20
21  print('fit: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_cv_score, recall_cv_score, f1_cv_score))
```

Latent Dirichlet allocation (LDA)

- LDA posits that documents are composed of a blend of various topics, with each topic being a collection of words.
- Given a corpus and a specified number of topics (k), LDA outputs a distribution of topics for each document.
- LDA can be seen as a method that discerns the variation in word distributions across different topics, thereby characterizing and separating these topics.

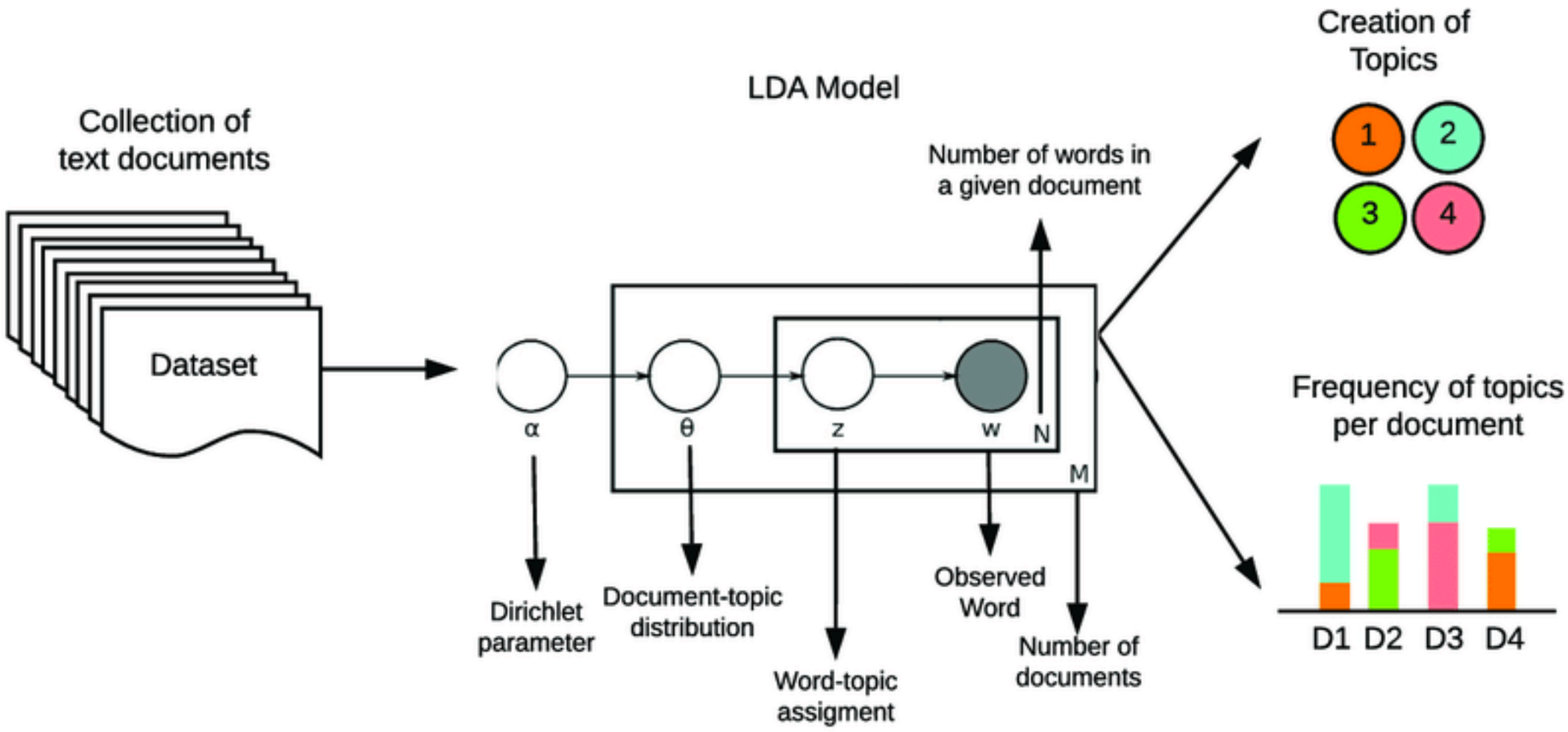
Latent Dirichlet allocation (LDA)





Ref: https://miro.medium.com/max/875/1*LsTAeih16RXl4VSZO_Hhag.png

Latent Dirichlet allocation (LDA)



Ref: Buenano-Fernandez, D., González, M., Gil, D., & Luján-Mora, S. (2020). Text mining of open-ended questions in self-assessment of university teachers: An LDA topic modeling approach. IEEE Access, 8

Latent dirichlet allocation (LDA)

```
01  from sklearn.decomposition import LatentDirichletAllocation
02
03  count_vectorizer = CountVectorizer(ngram_range=(1,1))
04  count_vectorizer.fit(cleaned_title + cleaned_body)
05  X_tf_fit = count_vectorizer.transform(data_fit['title'])
06  X_tf_blindtest = count_vectorizer.transform(data_blindtest['title'])
07  lda = LatentDirichletAllocation(n_components=500, random_state=0)
08  lda.fit(X_tf_fit)
08  X_lda_fit = lda.transform(X_tf_fit)
09  gbm_model_with_lda = lgb.LGBMClassifier()
10
11  precision_cv_score = model_selection.cross_val_score(gbm_model_with_lda, X_lda_fit, y_fit, cv=5,
    n_jobs=-2, scoring='precision_macro').mean()
12  recall_cv_score = model_selection.cross_val_score(gbm_model_with_lda, X_lda_fit, y_fit, cv=5,
    n_jobs=-2, scoring='recall_macro').mean()
13  f1_cv_score = model_selection.cross_val_score(gbm_model_with_lda, X_lda_fit, y_fit, cv=5, n_jobs=-2,
    scoring='f1_macro').mean()
14
15  print('fit: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_cv_score, recall_cv_score, f1_cv_score))
16
17  X_fit_with_lda = hstack([X_tfidf_fit, X_lda_fit]).tocsr()
18
19  precision_cv_score = model_selection.cross_val_score(gbm_model_with_lda, X_fit_with_lda, y_fit, cv=5,
    n_jobs=-2, scoring='precision_macro').mean()
20  recall_cv_score = model_selection.cross_val_score(gbm_model_with_lda, X_fit_with_lda, y_fit, cv=5,
    n_jobs=-2, scoring='recall_macro').mean()
21  f1_cv_score = model_selection.cross_val_score(gbm_model_with_lda, X_fit_with_lda, y_fit, cv=5,
    n_jobs=-2, scoring='f1_macro').mean()
22
23  print('fit: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_cv_score, recall_cv_score, f1_cv_score))
```

Performance?

- Just *tf-idf* CV
  - **precision:0.7513** recall: 0.7269 f1: 0.7292
- *tf-idf* + LSA CV
  - precision: 0.7400 recall: 0.7272 f1: 0.7295
- *tf-idf* + LDA CV
  - precision: 0.7472 **recall: 0.7302** **f1: 0.7326**

In class activity

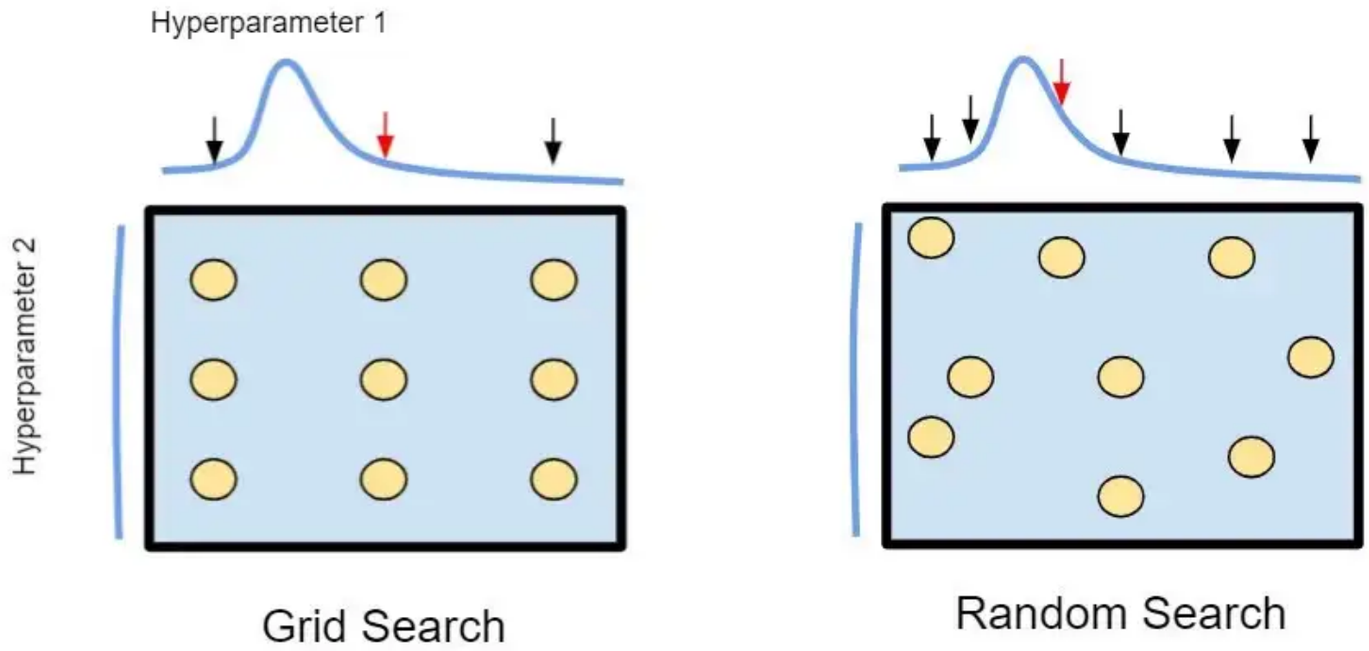
- Make a *tf-idf* + LSA + LDA version
- Please carefully design the dataflow
  - Raw data -> Tf-idf vectorizer -> basic … (1)
  - Raw data -> Tf-idf vectorizer -> LSA … (2)
  - Raw data -> Tf vectorizer -> LDA … (3)
  - GBM(1 + 2 + 3) -> predicted probability
- Make this *tf-idf* + LSA + LDA a flask application

Performance?

- Just *tf-idf* CV
  - **precision:0.7513** recall: 0.7269 f1: 0.7292
- *tf-idf* + LSA CV
  - precision: 0.7400 recall: 0.7272 f1: 0.7295
- *tf-idf* + LDA CV
  - precision: 0.7472 recall: 0.7302 **f1: 0.7326**
- *tf-idf* + LSA + LDA CV
  - precision: 0.7262 **recall: 0.7328** f1: 0.7279

Bonus topic — Parameter optimization

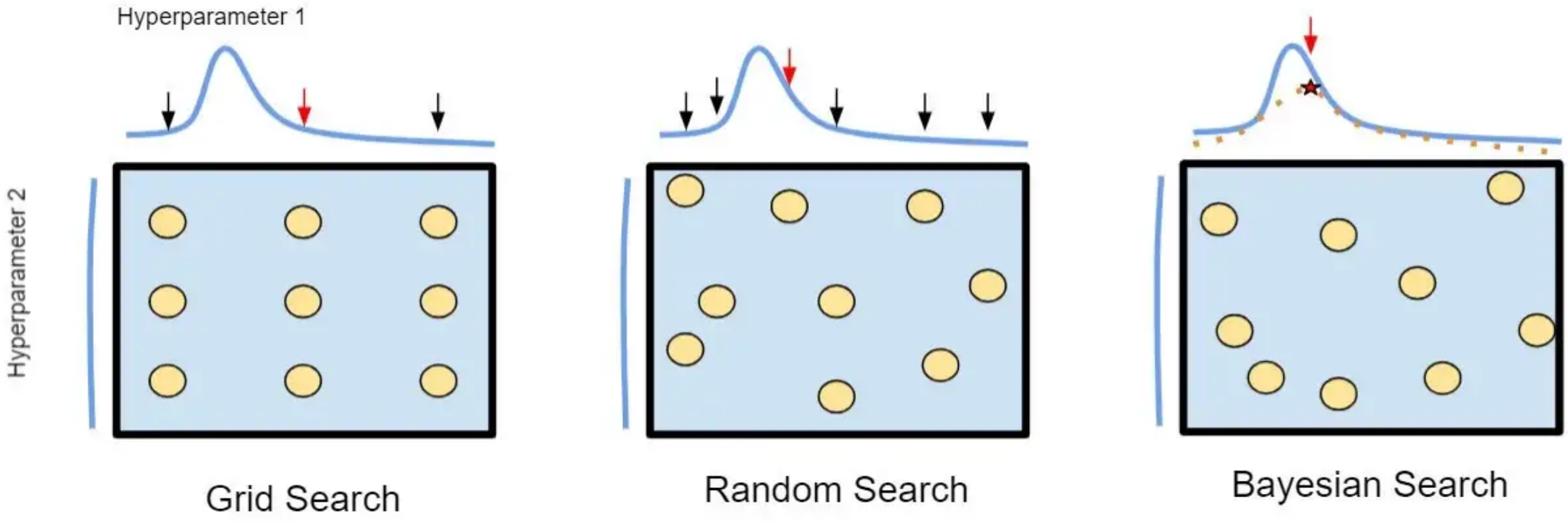
- Machine model have many parameters to configure, some may affect the overall performance.
- Common approaches are grid search and random search



Ref: https://towardsdatascience.com/hyperparameter-tuning-always-tune-your-models-7db7aeaf47e9

Bonus topic — Parameter optimization

- More recent techniques discussed Bayesian optimization
  - Can be considered as a guided search
  - May suffer by local optima but clearly better than grid search and random search



Ref: https://towardsdatascience.com/hyperparameter-tuning-always-tune-your-models-7db7aeaf47e9

Bayesian optimization

- Recommend python libraries
  - Optuna (https://optuna.org/)
  - Hyperopt (http://hyperopt.github.io/hyperopt/)
  - Ray (https://docs.ray.io/en/latest/tune/index.html)

Optuna example

```python
import optuna
def objective(trial):
    dtrain = lgb.Dataset(X_tfidf_fit_train, label=y_fit_train)

    param = {
        "objective": "binary",
        "metric": "binary_logloss",
        "verbosity": -1,
        "boosting_type": "gbdt",
        "lambda_l1": trial.suggest_float("lambda_l1", 1e-8, 10.0, log=True),
        "lambda_l2": trial.suggest_float("lambda_l2", 1e-8, 10.0, log=True),
        "num_leaves": trial.suggest_int("num_leaves", 2, 256),
        "feature_fraction": trial.suggest_float("feature_fraction", 0.4, 1.0),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.4, 1.0),
        "bagging_freq": trial.suggest_int("bagging_freq", 1, 7),
        "min_child_samples": trial.suggest_int("min_child_samples", 5, 100),
    }

    gbm = lgb.train(param, dtrain)
    preds = gbm.predict(X_tfidf_fit_test)
    pred_labels = np.rint(preds)
    accuracy = metrics.roc_auc_score(y_fit_test, pred_labels)
    return accuracy


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=30)
trial = study.best_trial
```

Optuna example

```
01  gbm_model = lgb.LGBMClassifier(**trial.params)
02
03  precision_cv_score = model_selection.cross_val_score(gbm_model, X_tfidf_fit, y_fit, cv=5,
    n_jobs=-2, scoring='precision_macro').mean()
04  recall_cv_score = model_selection.cross_val_score(gbm_model, X_tfidf_fit, y_fit, cv=5, n_jobs=-2,
    scoring='recall_macro').mean()
05  f1_cv_score = model_selection.cross_val_score(gbm_model, X_tfidf_fit, y_fit, cv=5, n_jobs=-2,
    scoring='f1_macro').mean()
06
07  print('CV: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_cv_score, recall_cv_score,
    f1_cv_score))
08
09  gbm_model.fit(X_tfidf_fit_train, y_fit_train, eval_set=[(X_tfidf_fit_test, y_fit_test)],
    eval_metric='AUC')
10
11  precision_test_score = metrics.precision_score(gbm_model.predict(X_tfidf_blindtest), y_blindtest,
    average='macro')
12  recall_test_score = metrics.recall_score(gbm_model.predict(X_tfidf_blindtest), y_blindtest,
    average='macro')
13  f1_test_score = metrics.f1_score(gbm_model.predict(X_tfidf_blindtest), y_blindtest,
    average='macro')
14
15  print('test: p:{0:.4f} r:{1:.4f} f:{2:.4f}'.format(precision_test_score, recall_test_score,
    f1_test_score))
```

Performance?

- Just tf-idf CV
  - precision:0.7495 recall: 0.7277 f1: 0.7301
- tf-idf + LDA CV
  - precision: 0.7485 recall: 0.7327 f1: 0.7351
- Tf-idf + Optuna CV
  - precision: 0.7508 recall: 0.7396 f1: 0.7419

In class activity

- Make an Optuna optimized *tf-idf* + LDA version
- Please carefully design the dataflow
  - Raw data -> Tf-idf vectorizer -> basic … (1)
  - Raw data -> Tf vectorizer -> LDA … (2)
  - GBM(1 + 2) -> predicted probability
- Make this Optuna optimized *tf-idf* + LDA a flask application

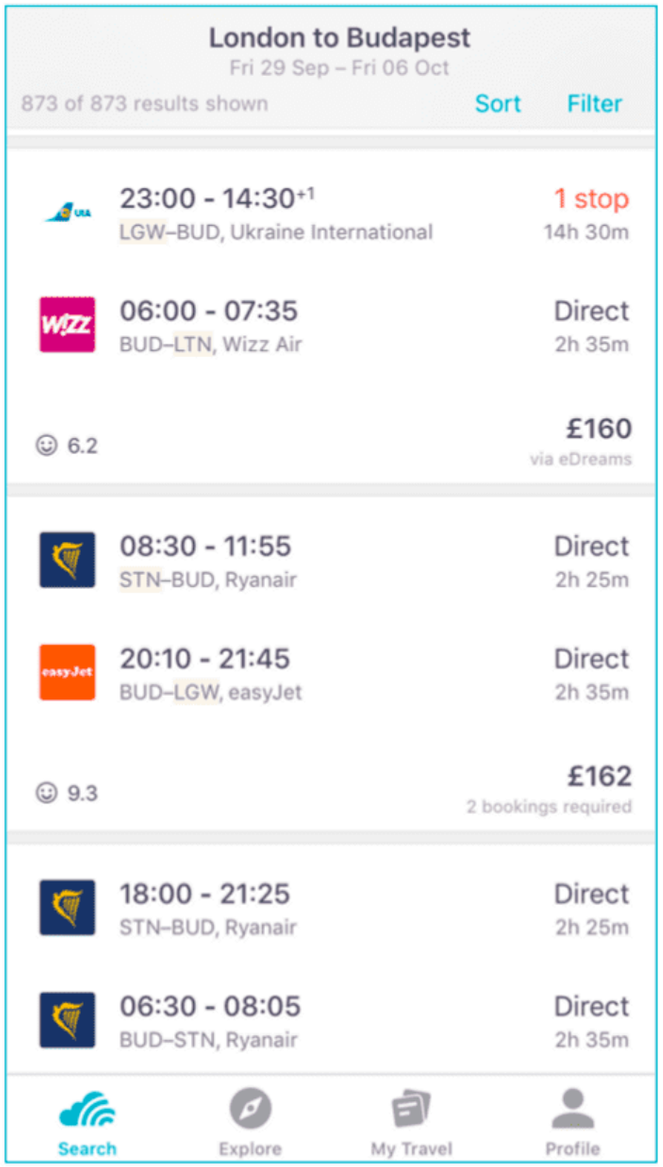
Performance?

- Just tf-idf CV
  - precision:0.7495 recall: 0.7277 f1: 0.7301
- tf-idf + LDA CV
  - precision: 0.7485 recall: 0.7327 f1: 0.7351
- Tf-idf + Optuna CV
  - **precision: 0.7508** recall: 0.7396 f1: 0.7419
- **Tf-idf + LDA + Optuna CV**
  - precision: 0.7503 **recall: 0.7437** **f1: 0.7455**

Applying ML to IR — Learning to rank

- Ranking is the process of ordering documents by **their relevance** to a given **query**, aiming to surface the most pertinent content.
- Machine learning algorithms can **quantify relevance**, simplifying the complex task of scoring document relevance.
- These algorithms are designed to optimize ranking performance, typically measured by metrics like mean average precision (**mAP**) or normalized discounted cumulative gain (**nDCG**).

Applying ML to IR — Learning to rank



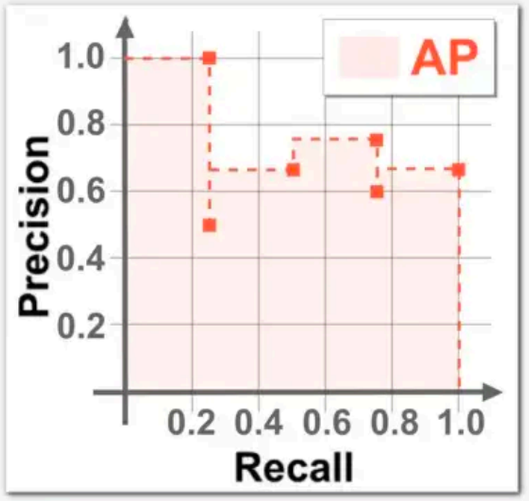
You saw this, but you didn't buy it.

You saw this, but you didn't buy it.

You saw this, and bought it.

Ref: https://lucidworks.com/post/abcs-learning-to-rank/

Applying ML to IR — Learning to rank



$$\mathrm{AP} = \sum_{k=1}^{n} (\mathrm{R}_k - \mathrm{R}_{k-1})\,\mathrm{P}_k$$

	1st	2nd	3rd	4th	5th	6th	
y_k	1✓	0✗	1✓	1✓	0✗	1✓	
P_k	1.00	0.50	0.67	0.75	0.60	0.67	AP= 0.77
R_k	0.25	0.25	0.50	0.75	0.75	1.00	

$$\mathrm{G}_k = 2^{y_k} - 1$$

$$\mathrm{D}_k = \frac{1}{\log_2(k+1)}$$

$$\mathrm{DCG} = \sum_{k=1}^{n} \mathrm{G}_k \mathrm{D}_k$$

	1st	2nd	3rd	4th	5th	6th	
y_k	3	4	1	2	0	1	
G_k	7.00	15.00	1.00	3.00	0.00	1.00	DCG 18.61
D_k	1.00	0.63	0.50	0.43	0.39	0.36	NDCG 0.86

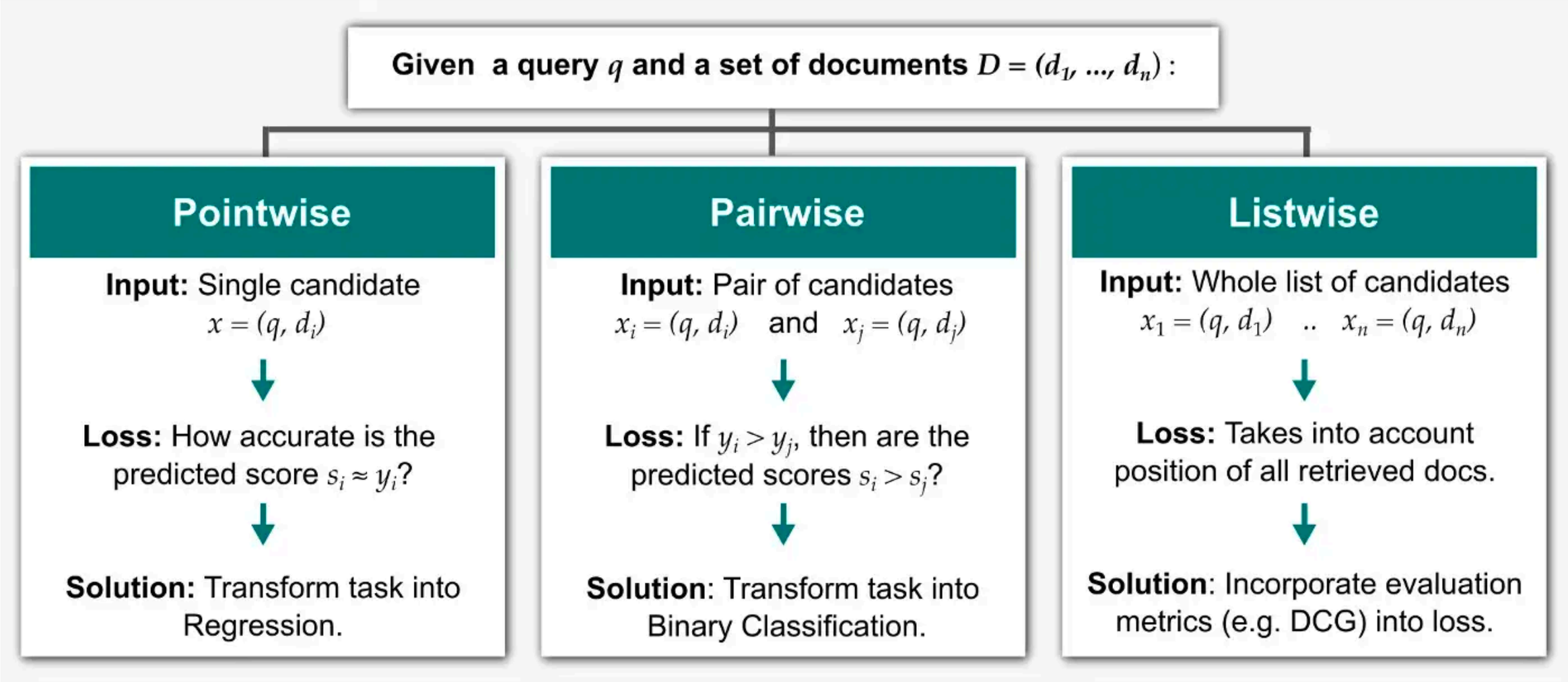
Ref: https://towardsdatascience.com/learning-to-rank-a-complete-guide-to-ranking-using-machine-learning-4c9688d370d4

Applying ML to IR — Learning to rank

- **Input** — For a given query q, a set of $D = \{d_1, ..., d_n\}$ is considered. The pairs $x_i = (q, d_i)$ serve as the inputs for the ranking model

- **Output** — The model predicts a score $s_i = f(x_i)$ for each query-document pair $x_i = (q, d_i)$, representing its relevance. This score is intended to approximate the true relevance score y_i, with the model's objective being to minimize the discrepancy between s_i and y_i.

What is applied to IR — Learning to rank



Ref: https://towardsdatascience.com/learning-to-rank-a-complete-guide-to-ranking-using-machine-learning-4c9688d370d4

Learning to rank — an example

- Try with an easy-to-access algorithm
  - lightgbm.LGBMRanker implementation of LambdaRank
  - In part of LightGBM (Back by Microsoft)
  - Not fully compatible with SK-learn as it doesn’t support ranking applications yet at the time of writing

- A motivational example
  - An anime recommender — https://www.kaggle.com/code/ekity1002/lightgbm-learning-to-rank-anime-recommender

Learning to rank example

```
anime = pd.read_csv('resource/anime.csv')
rating = pd.read_csv('resource/anime_rating_1000_users.csv')
```

```
anime_features = ['MAL_ID','English name','Japanese name','Score','Genres','Popularity',
            'Members','Favorites','Watching','Completed','On-Hold','Dropped',
            'Score-1','Score-2','Score-3','Score-4','Score-5',
            'Score-6','Score-7','Score-8','Score-9','Score-10',
            ]
anime = anime[anime_features]

#%%

merged_df = anime.merge(rating, left_on='MAL_ID', right_on='anime_id', how='inner')

#%%

genre_names = [
    'Action', 'Adventure','Comedy', 'Drama','Sci-Fi',
    'Game', 'Space', 'Music', 'Mystery', 'School', 'Fantasy',
    'Horror', 'Kids', 'Sports', 'Magic', 'Romance',
]
```

Learning to rank example

```python
def genre_to_category(df):
    '''Add genre cagegory column
    '''
    d = {name :[] for name in genre_names}

    def f(row):
        genres = row.Genres.split(',')
        for genre in genre_names:
            if genre in genres:
                d[genre].append(1)
            else:
                d[genre].append(0)

    # create genre category dict
    df.apply(f, axis=1)

    # add genre category
    genre_df = pd.DataFrame(d, columns=genre_names)
    df = pd.concat([df, genre_df], axis=1)
    return df

def make_anime_feature(df):
    # convert object to a numeric type, replacing Unknown with nan.
    df['Score'] = df['Score'].apply(lambda x: np.nan if x=='Unknown' else float(x))
    for i in range(1, 11):
        df[f'Score-{i}'] = df[f'Score-{i}'].apply(lambda x: np.nan if x=='Unknown' else float(x))

    # add genre ctegory columns
    df = genre_to_category(df)

    return df
```

Learning to rank example

```python
def make_user_feature(df):
    # add user feature
    df['rating_count'] = df.groupby('user_id')['anime_id'].transform('count')
    df['rating_mean'] = df.groupby('user_id')['rating'].transform('mean')
    return df

def preprocess(merged_df):
    merged_df = make_anime_feature(merged_df)
    merged_df = make_user_feature(merged_df)
    return merged_df
```

```python
merged_df = preprocess(merged_df)
merged_df = merged_df.drop(['MAL_ID', 'Genres'], axis=1)
```

Learning to rank example

```
fit, blindtest = train_test_split(merged_df, test_size=0.2, random_state=0)
fit_train, fit_test = train_test_split(fit, test_size=0.3, random_state=0)

features = ['Score', 'Popularity','Members',
            'Favorites','Watching','Completed','On-Hold','Dropped',
            'Score-1','Score-2','Score-3','Score-4','Score-5',
            'Score-6','Score-7','Score-8','Score-9','Score-10',
            'rating_count','rating_mean'
           ]
features += genre_names
user_col = 'user_id'
item_col = 'anime_id'
target_col = 'rating'

fit_train = fit_train.sort_values('user_id').reset_index(drop=True)
fit_test = fit_test.sort_values('user_id').reset_index(drop=True)
blindtest = blindtest.sort_values('user_id').reset_index(drop=True)

# model query data
fit_train_query = fit_train[user_col].value_counts().sort_index()
fit_test_query = fit_test[user_col].value_counts().sort_index()
blindtest_query = blindtest[user_col].value_counts().sort_index()
```

Learning to rank example

```
model = lgb.LGBMRanker(n_estimators=1000, random_state=0)
model.fit(
    fit_train[features],
    fit_train[target_col],
    group=fit_train_query,
    eval_set=[(fit_test[features], fit_test[target_col])],
    eval_group=[list(fit_test_query)],
    eval_at=[1, 3, 5, 10], # calc validation ndcg@1,3,5,10
    early_stopping_rounds=100,
    verbose=10
)

#%%

model.predict(blindtest.iloc[:10][features])
```

Learning to rank example

```
1   # feature imporance
2   plt.figure(figsize=(10, 7))
3   df_plt = pd.DataFrame({'feature_name': features, 'feature_importance':
    model.feature_importances_})
4   df_plt.sort_values('feature_importance', ascending=False, inplace=True)
5   sns.barplot(x="feature_importance", y="feature_name", data=df_plt)
6   plt.title('feature importance')
```

Learning to rank example

```python
def predict(user_df, top_k, anime, rating):
    user_anime_df = anime.merge(user_df, left_on='MAL_ID', right_on='anime_id')
    user_anime_df = make_anime_feature(user_anime_df)

    excludes_genres = list(np.array(genre_names)[np.nonzero([user_anime_df[genre_names].sum(axis=0) <= 1])[1]])

    pred_df = make_anime_feature(anime.copy())
    pred_df = pred_df.loc[pred_df[excludes_genres].sum(axis=1)==0]

    for col in user_df.columns:
        if col in features:
            pred_df[col] = user_df[col].values[0]

    preds = model.predict(pred_df[features])

    topk_idx = np.argsort(preds)[::-1][:top_k]

    recommend_df = pred_df.iloc[topk_idx].reset_index(drop=True)

    # check recommend
    print('---------- Recommend ----------')
    for i, row in recommend_df.iterrows():
        print(f'{i+1}: {row["Japanese name"]}:{row["English name"]}')

    print('---------- Rated ----------')
    user_df = user_df.merge(anime, left_on='anime_id', right_on='MAL_ID', how='inner')
    for i, row in user_df.sort_values('rating',ascending=False).iterrows():
        print(f'rating:{row["rating"]}: {row["Japanese name"]}:{row["English name"]}')

    return recommend_df
```

Learning to rank example

```
01  user_df = rating.copy().loc[rating['user_id'] == user_id]
02  user_df = make_user_feature(user_df)
03  predict(user_df, 10, anime, rating)
```

Assignment

- Make an **Optuna** optimized ***tf-idf*** **+ LSA + LDA** that taking into an account for both **title** and **body**
  - Try to find-tune to get the highest CV accuracy possible;
  - Submit the iPython notebook show the final accuracy;
  - **Note:** in addition, there are rooms to improve the **preprocess**, **parameter optimization.**
- Finally, integrate your best configured integration into the flask application

Time for questions