Lecture Information

Course: CSCI E-89B: Natural Language Processing
Lecture: Lecture 9
Topic: Classical Machine Learning for NLP
Date: Fall 2024

Contents

1 Introduction to Classical ML for NLP

Overview

This lecture covers classical machine learning techniques applied to natural language processing. While neural networks dominate modern NLP, classical methods remain valuable for smaller datasets and provide interpretable baselines.

1.1 Why Classical Methods?

When to Use Classical Methods

Classical methods are particularly useful when:

- You have a **small dataset** (neural networks need lots of data)
- You need **interpretability** (understand why predictions are made)
- You need **fast training** (no GPU required)
- You want a **baseline** before trying complex models

1.2 Text Representation for Classical Methods

Important

Classical methods require **numerical vectors** as input. We cannot feed raw text directly. Common representations:

- **TF-IDF vectors**: Term frequency-inverse document frequency
- **Bag of Words**: Simple word counts
- **N-gram features**: Sequences of consecutive words

2 Naive Bayes Classifier

Overview

Naive Bayes is a probabilistic classifier based on Bayes' theorem with a "naive" assumption of feature independence. Despite its simplicity, it works surprisingly well for text classification.

2.1 Bayes' Theorem

Bayes' Theorem

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

For classification:

$$P(C_k|X) = \frac{P(X|C_k) \cdot P(C_k)}{P(X)}$$

where:

- C_k: Class k (e.g., spam or not spam)
- X: Feature vector (e.g., TF-IDF representation)
- $P(C_k)$: Prior probability of class k
- $P(X|C_k)$: Likelihood of features given class
- $P(C_k|X)$: Posterior probability (what we want)

2.2 The Naive Assumption

Conditional Independence

Naive Bayes assumes features are **conditionally independent** given the class:

$$P(X|C_k) = P(x_1|C_k) \cdot P(x_2|C_k) \cdot \ldots \cdot P(x_n|C_k) = \prod_{i=1}^{n} P(x_i|C_k)$$

Why "Naive"?

The independence assumption is rarely true in practice:

- If "cat" appears, "dog" is less likely in the same document
- Words are correlated through context

However, TF-IDF already loses much contextual information, so the assumption is less problematic than it seems. The classifier still performs well in practice!

2.3 Classification Decision

Classification Rule

Choose the class with highest posterior probability:

$$\hat{y} = \arg\max_{C_k} P(C_k|X) = \arg\max_{C_k} P(C_k) \prod_{i=1}^{n} P(x_i|C_k)$$

In practice, use log probabilities for numerical stability:

$$\hat{y} = \arg\max_{C_k} \left[\log P(C_k) + \sum_{i=1}^{n} \log P(x_i|C_k) \right]$$

2.4 Applications

Common Applications

- **Spam detection**: Classic application
- **Sentiment analysis**: Positive/negative classification
- **Document categorization**: News topic classification

2.5 Implementation

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

# Sample documents
documents = [
    "Cats are wonderful pets",
    "Dogs enjoy long walks",
    # ... more documents
]
labels = [0, 1, ...]  # 0=cats, 1=dogs

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    documents, labels, test_size=0.25
)

# Create TF-IDF vectors
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)  # Don't refit!

# Train and predict
nb = MultinomialNB()
nb.fit(X_train_tfidf, y_train)
predictions = nb.predict(X_test_tfidf)
```

Listing 1: Naive Bayes for Text Classification

Data Leakage Warning

Always fit the TF-IDF vectorizer on **training data only**, then transform test data. Never fit on test data—this causes data leakage!

3 K-Nearest Neighbors (KNN)

Overview

KNN is a simple, non-parametric algorithm that classifies based on the labels of the K closest training examples. It requires no training phase—it memorizes the entire dataset.

3.1 How KNN Works

KNN Algorithm

For a new data point:

1. Compute distances to all training points
2. Find the K nearest neighbors
3. **Classification**: Take majority vote among K neighbors
4. **Regression**: Take average of K neighbors' values

1D Example

Data points: $\{(1,0),(2,0),(3,0),(5,1),(6,1),(7,1)\}$
To classify point $x = 4$ with $K = 3$:

1. Nearest neighbors: $x = 3, 5, 6$ (or $3, 5, 2$)
2. Labels: $\{0, 1, 1\}$
3. Majority vote: 1 wins
4. Prediction: Class 1

3.2 Distance Metrics

Common Distance Metrics

Euclidean Distance (default):

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Manhattan Distance:

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

Cosine Similarity (often better for text):

$$\text{similarity}(x,y) = \frac{x \cdot y}{\|x\|\|y\|}$$

3.3 Feature Scaling

Scale Your Features!

If features are on different scales, larger-scale features dominate:

- Age: 20–80 years (range: 60)
- Income: \$20,000–\$200,000 (range: 180,000)

Income will completely dominate the distance calculation! Always normalize features to similar scales.

3.4 Choosing K

Important

Use odd K for binary classification to avoid ties.
Choosing K:

- Too small K: Overfitting (sensitive to noise)
- Too large K: Underfitting (ignores local structure)
- Optimize K using validation set or cross-validation

3.5 Is KNN Deterministic?

KNN Can Be Stochastic!

KNN is **not always deterministic**:

- **Even K**: May have ties in majority vote
- **Equal distances**: Multiple points at same distance—which to include?

Ties must be broken randomly, so results may vary between runs. Default implementations often run multiple times and average.

3.6 Implementation

```python
from sklearn.neighbors import KNeighborsClassifier

# After TF-IDF vectorization...
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_tfidf, y_train)
predictions = knn.predict(X_test_tfidf)

# Optimize K
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1, 21)}
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_tfidf, y_train)
print(f"Best K: {grid_search.best_params_}")
```

Listing 2: KNN for Text Classification

4 Logistic Regression

Overview

Logistic regression is a linear model for binary classification. It's equivalent to a single-layer neural network with sigmoid activation.

4.1 The Model

Logistic Regression

$$P(y = 1|X) = \sigma(W \cdot X + b) = \frac{1}{1 + e^{-(W \cdot X + b)}}$$

where:

- σ: Sigmoid function
- W: Weight vector (learned)
- b: Bias term (learned)
- X: Feature vector

4.2 Decision Boundary

Important

The decision boundary is where $P(y = 1|X) = 0.5$, which means $W \cdot X + b = 0$.
In 2D, this is a **straight line**. In higher dimensions, it's a **hyperplane**.
Limitation: Logistic regression can only separate classes that are linearly separable.

4.3 Connection to Neural Networks

Logistic Regression

Logistic regression is exactly a neural network with:

- No hidden layers
- Sigmoid activation on output
- Binary cross-entropy loss

Training via maximum likelihood is mathematically equivalent to minimizing cross-entropy loss.

4.4 Non-linear Decision Boundaries

Feature Engineering for Non-linearity

If data isn't linearly separable, add polynomial features:
Original features: x_1, x_2
Extended features: $x_1, x_2, x_1^2, x_2^2, x_1 x_2$
This allows curved decision boundaries, but requires manual feature design.

4.5 Implementation

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_tfidf, y_train)
predictions = lr.predict(X_test_tfidf)
```

```
6
7 # Get probabilities
8 probabilities = lr.predict_proba(X_test_tfidf)
```

Listing 3: Logistic Regression for Text

5 Support Vector Machines (SVM)

Overview

SVMs were the dominant machine learning method before deep learning (pre-2012). They find the hyperplane that maximizes the margin between classes.

5.1 The Maximum Margin Idea

Maximum Margin Classifier

Given two linearly separable classes, find the hyperplane that:

- Correctly separates all points
- Maximizes the **margin**—the distance to the nearest points (support vectors)

Visualizing the Margin

Imagine two parallel lines (in 2D) separating two classes. The gap between these lines is the margin. SVM finds the position that makes this gap as wide as possible.
Points touching the margin boundaries are **support vectors**—only these points determine the decision boundary position.

5.2 Mathematical Formulation

SVM Optimization

The hyperplane is defined by $W \cdot X + b = 0$.
Hard Margin SVM (linearly separable):

$$\min_{W,b} \frac{1}{2}\|W\|^2 \quad \text{subject to} \quad y_i(W \cdot X_i + b) \geq 1 \quad \forall i$$

This maximizes the margin $\frac{2}{\|W\|}$.

5.3 The Overfitting Problem

Hard Margin Overfitting

With hard margin SVM, only support vectors affect the decision boundary. Moving one support vector changes everything!
This is **overfitting**—the model is too sensitive to individual points.

5.4 Soft Margin SVM

Soft Margin SVM

Allow some points to be inside the margin or misclassified:

$$\min_{W,b,\xi} \frac{1}{2}\|W\|^2 + C\sum_i \xi_i \quad \text{subject to} \quad y_i(W \cdot X_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

where:

- ξ_i: Slack variable (how far point i is from correct side)
- C: Regularization parameter (trade-off between margin and violations)

Important

Parameter C:

- Large C: Few violations allowed, narrower margin (risk overfitting)
- Small C: More violations allowed, wider margin (risk underfitting)

Tune C using cross-validation.

5.5 Non-linear Decision Boundaries: Kernels

What if data isn't linearly separable?

Consider points arranged in a circle (class 0) surrounded by points outside (class 1). No line can separate them!

The Kernel Trick

Add a new dimension: $z = x_1^2 + x_2^2$ (distance from origin).
In this 3D space, points are linearly separable by a plane!
Key insight: We don't actually compute the transformation. Instead, we replace the dot product $X_i \cdot X_j$ with a **kernel function** $K(X_i, X_j)$.

5.6 Common Kernels

Kernel	Formula
Linear	$K(x, y) = x \cdot y$
Polynomial	$K(x, y) = (x \cdot y + c)^d$
RBF (Gaussian)	$K(x, y) = \exp(-\gamma\|x - y\|^2)$

RBF Kernel

The RBF (Radial Basis Function) kernel can approximate any decision boundary. It's equivalent to projecting to infinite dimensions!
The γ parameter controls how "local" the influence of each point is.

5.7 Implementation

```python
from sklearn.svm import SVC

# Linear kernel
svm_linear = SVC(kernel='linear', C=1.0)
svm_linear.fit(X_train_tfidf, y_train)

# RBF kernel
svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_rbf.fit(X_train_tfidf, y_train)

# Grid search for optimal parameters
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly']
}
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train_tfidf, y_train)
```

Listing 4: SVM for Text Classification

6 Decision Trees and Random Forests

Overview

Decision trees recursively split data based on feature values. Random forests combine many trees to reduce overfitting and improve accuracy.

6.1 Decision Tree Algorithm

Building a Decision Tree

At each node:

1. Consider all features and all possible split points
2. Choose the split that best separates classes
3. Recurse on each resulting subset
4. Stop when a criterion is met (max depth, min samples, pure node)

6.2 Splitting Criteria

Gini Index (Classification)

For a node with proportion p of class 1:

$$\text{Gini} = 1 - (p^2 + (1-p)^2) = 2p(1-p)$$

- Pure node ($p = 0$ or $p = 1$): Gini $= 0$
- Maximum impurity ($p = 0.5$): Gini $= 0.5$

Choose splits that minimize total Gini index of child nodes.

Entropy (Alternative)

$$\text{Entropy} = -p\log_2(p) - (1-p)\log_2(1-p)$$

Information gain = parent entropy minus weighted child entropy.

6.3 Overfitting in Decision Trees

Trees Easily Overfit

Without restrictions, a decision tree will:

- Split until every leaf contains one sample
- Perfectly classify training data
- Perform poorly on test data

Solutions: Limit max depth, require minimum samples per leaf, or pruning.

6.4 Random Forests

Random Forest Algorithm

1. Create B bootstrap samples (sample with replacement)
2. For each sample, build a decision tree:
   - At each split, consider only $\sqrt{n}$ random features
   - Grow tree fully (no pruning needed)
3. Final prediction: Majority vote (classification) or average (regression)

6.5 Why Random Forests Work

Important

Two sources of randomness:

1. **Bootstrap sampling**: Each tree sees different data
2. **Random feature selection**: Each split considers different features

This creates **diverse** trees. When averaged, individual tree errors cancel out, leaving only the "signal."
Random forests are nearly **impossible to overfit**!

6.6 Bootstrap Sampling

Bootstrap Sample

Sample n points **with replacement** from dataset of size n:

- Some points appear multiple times
- Some points don't appear at all (37%)
- Each bootstrap sample is slightly different

6.7 Interpreting Black Box Models

Interpretability Challenge

Random forests are "black boxes"—hard to understand why predictions are made.
Solutions:

- **Partial Dependence Plots (PDP)**: Show how each feature affects predictions
- **SHAP values**: Explain individual predictions
- **Feature importance**: Which features matter most

6.8 Implementation

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=100,        # Number of trees
    max_features='sqrt',     # Features per split
    criterion='gini',        # Splitting criterion
    random_state=42
)
rf.fit(X_train_tfidf, y_train)
predictions = rf.predict(X_test_tfidf)

# Feature importance
importance = rf.feature_importances_
```

Listing 5: Random Forest for Text

7 K-Fold Cross-Validation

Overview

K-fold cross-validation is essential for classical methods where datasets are small and we need reliable performance estimates.

7.1 Why Cross-Validation?

Important

For neural networks with large datasets:

- Split into train/validation/test
- Plenty of data for each split

For classical methods with small datasets:

- Can't afford to reserve much for testing
- Single split may not be representative
- Need K-fold cross-validation

7.2 K-Fold Procedure

K-Fold Cross-Validation

1. Split data into K equal parts (folds)
2. For $i = 1, \ldots, K$:
   - Use fold i as test set
   - Use remaining $K - 1$ folds as training set
   - Train model and evaluate on fold i
3. Average performance across all K folds

5-Fold Cross-Validation

Data split: [Fold 1] [Fold 2] [Fold 3] [Fold 4] [Fold 5]

- Run 1: Train on 2,3,4,5, test on 1
- Run 2: Train on 1,3,4,5, test on 2
- Run 3: Train on 1,2,4,5, test on 3
- Run 4: Train on 1,2,3,5, test on 4
- Run 5: Train on 1,2,3,4, test on 5

Every data point is tested exactly once!

7.3 Using Pipelines

TF-IDF Must Be Refit Each Fold!

When using K-fold, the TF-IDF vectorizer must be fit on training folds only, not test fold. Use sklearn Pipeline to handle this automatically.

```
1 from sklearn.pipeline import Pipeline
```

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold

# Create pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english')),
    ('clf', MultinomialNB())
])

# K-fold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(pipeline, documents, labels, cv=cv, scoring='accuracy')

print(f"Mean accuracy: {scores.mean():.4f}")
print(f"Std: {scores.std():.4f}")
```

Listing 6: Pipeline with Cross-Validation

8 Evaluation Metrics for Classification

8.1 Confusion Matrix

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

8.2 Key Metrics

Classification Metrics

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity (Recall, True Positive Rate):

$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{\text{Correctly predicted positives}}{\text{All actual positives}}$$

Specificity (True Negative Rate):

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{\text{Correctly predicted negatives}}{\text{All actual negatives}}$$

Precision:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{\text{True positives}}{\text{All predicted positives}}$$

F1 Score:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

8.3 Imbalanced Datasets

Accuracy is Misleading for Imbalanced Data

If 95% of emails are not spam:

- Always predicting "not spam" gives 95% accuracy!
- But sensitivity for spam is 0%

Solutions:

- Use precision/recall/F1 instead of accuracy
- Adjust classification threshold (from 0.5)
- Use class weights or focal loss
- Resample data (SMOTE, undersampling)

9 Model Comparison: Text Classification Example

BBC News Classification Results

Task: Classify news articles as "tech" or "not tech"

Model	Accuracy	Sensitivity	Specificity
Naive Bayes	0.90	0.43	1.00
KNN (K=5)	0.97	0.94	0.98
KNN (K=19, optimized)	0.98	0.94	0.99
Logistic Regression	0.92	0.71	1.00
SVM (linear)	0.92	0.73	1.00
SVM (optimized)	0.99	0.97	1.00
Random Forest	0.98	0.89	1.00

Winner: SVM with optimized parameters (C=10, linear kernel)

Observations

- Models predicting mostly "not tech" have high specificity but low sensitivity
- This happens because "not tech" is the majority class
- SVM (pre-neural network era champion) performs best overall
- KNN performs surprisingly well with optimized K

10 One-Page Summary

Summary

Classical ML for NLP: Still valuable for small datasets and interpretability.
Naive Bayes:

- Assumes feature independence: $P(X|C) = \prod_i P(x_i|C)$
- Fast, simple, works well for text
- Use: Spam detection, sentiment analysis

K-Nearest Neighbors:

- Classify by majority vote of K nearest points
- No training—stores entire dataset
- Scale features! Choose odd K
- Tune K via cross-validation

Logistic Regression:

- $P(y = 1|X) = \sigma(W \cdot X + b)$
- Linear decision boundary (hyperplane)
- Equivalent to single-layer neural network

Support Vector Machines:

- Maximize margin between classes
- Soft margin allows violations (parameter C)
- Kernels enable non-linear boundaries (RBF, polynomial)
- Dominant method pre-2012

Random Forests:

- Ensemble of decision trees
- Bootstrap + random feature selection
- Nearly impossible to overfit
- Interpret via SHAP, PDP

K-Fold Cross-Validation:

- Essential for small datasets
- Each point tested exactly once
- Use Pipeline to refit TF-IDF each fold

Metrics for Imbalanced Data:

- Accuracy misleading when classes unbalanced
- Use precision, recall, F1, AUC-PR

11 Glossary

Key Terms

- **Naive Bayes**: Probabilistic classifier assuming feature independence
- **KNN**: K-Nearest Neighbors—classify by neighbor majority
- **Logistic Regression**: Linear model with sigmoid output
- **SVM**: Support Vector Machine—maximize margin classifier
- **Kernel**: Function replacing dot product to enable non-linear boundaries
- **Support Vector**: Point touching the margin boundary
- **Soft Margin**: SVM allowing margin violations
- **Decision Tree**: Recursive feature splitting
- **Random Forest**: Ensemble of trees with bootstrap + random features
- **Bagging**: Bootstrap AGGregatING—averaging bootstrap models
- **Gini Index**: Impurity measure for splitting
- **Cross-Validation**: Rotating train/test splits
- **Sensitivity (Recall)**: TP / (TP + FN)
- **Specificity**: TN / (TN + FP)
- **Precision**: TP / (TP + FP)
- **F1 Score**: Harmonic mean of precision and recall
- **Data Leakage**: Using test information during training
- **Bootstrap Sample**: Sample with replacement
- **PDP**: Partial Dependence Plot for interpretability
- **SHAP**: Shapley Additive Explanations for feature importance