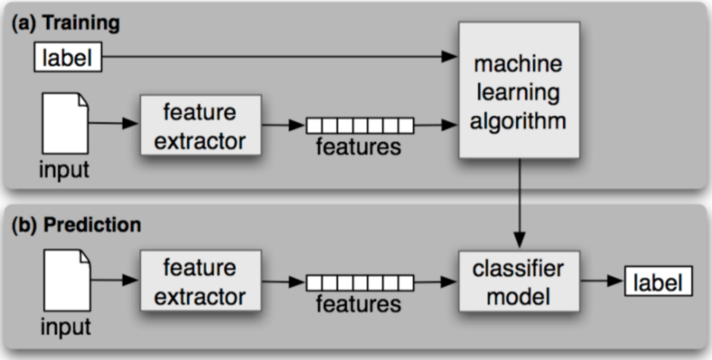
Maximum Entropy Klassifikator; Klassifikation mit Scikit-Learn

Benjamin Roth

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilian-Universität München
beroth@cis.uni-muenchen.de

Classification



- We learned about Naive-Bayes and Perceptron.
- What other common classification methods are there?
- How to use them in Scikit-Learn?

MaxEnt Model (Theory)

Introduction to Maximum Entropy (MaxEnt) Model

- Generative models like Naive Bayes classifiers place the probabilities over both observed data and the class $\Rightarrow P(c, d)$
- Discriminative models take the data as given and put the probability over the hidden class $\Rightarrow P(c|d)$
  - Maximum entropy model
  - Support Vector Machine (SVM)

Introduction to MaxEnt Model

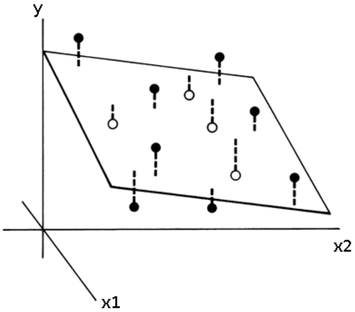
- Discriminative models work better than generative models in many tasks
- Word sense disambiguation, Klein and Manning 2002
  - Classify the sense of a word depending on the context
  - Exactly the same features
  - Naive Bayes: 73.6% accuracy
  - MaxEnt: 76.1% accuracy
- Text classification on Reuter dataset, Zhang and Oles, 2001
  - Features are words in documents and class
  - Naive Bayes: 77.0% F-score
  - MaxEnt: 86.4% F-score
  - Emphasizes the importance of regularization ("smoothing" for discriminative models)

What is the Maximum Entropy Model?

- ... a linear regression model turned into a probability distribution $P(class|features)$.
- First: predict score z_i for each outcome i
  - Multiply each feature value with corresponding weight
  - Scores can be $\in]-\infty, \infty[$
  - Outcomes depend on task, e.g. {ham,spam} or {Noun, Verb, Adj, ...} or {positive, negative, neutral}
- Then: Rescale and normalize
  - Probability must be positive for all outcomes
  - Probabilities for outcomes must sum up to 1
  - $\Rightarrow$ exponentiate and normalize $\boldsymbol{z}$
- Learning the feature weights: maximize probability of training set (*maximum likelihood estimation*)
- Other names:
  - logistic regression, logit-model, log-linear model

Why called MaxEnt?
Yields the model with maximum entropy, where the expectation of feature observations equals the observed feature values.

Linear Regression

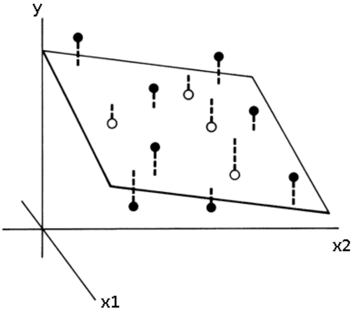


- Linear function:

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} = \sum_{j=1}^{n} w_j x_j$$

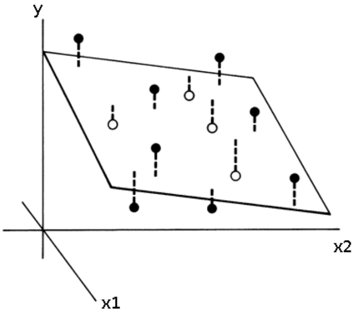
- Parameter vector $\boldsymbol{w} \in \mathbb{R}^n$
  Weight w_j decides if value of feature x_j increases or decreases prediction $\hat{y}$.

Linear Regression



- Example: Levitt and Dubner (2005) showed that the words used in real estate ads can be used to predict housing price
- "fantastic", "cute" or "charming" $\Rightarrow$ lower price
- "maple", "granite" $\Rightarrow$ higher price
- $price = w_0 + w_1 \cdot Num\_Adjectives + w_2 \cdot Mortgage\_Rate + w3 \cdot Num\_Unsold\_Houses$
- Would w_1 be positive or negative?

Back to MaxEnt



- In the maximum entropy model we want to use regression for classification.
- If you have k classes to predict, use regression to predict one score for each class.
- How many feature weights?
- How to write multiplication of design matrix with feature weights?

Back to MaxEnt

- How many feature weights?
  There is one weight for each combination of all features with all classes.
- How to write multiplication of design matrix with feature weights?
  If there are n instances, m features and k classes, then the design matrix $\boldsymbol{X}$ is $n \times m$ and the weight matrix $\boldsymbol{W}$ is often set to be $k \times m$. The matrix containing all scores for all instances is $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}^T$

Score prediction for one instance

- k classes and m features
- m-dimensional feature vector for one instance: $\boldsymbol{x}$
- Weight matrix $\boldsymbol{W}$ has shape $k \times m$
- $$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix} = \boldsymbol{W}\boldsymbol{x}$$

  (i.e. $\boldsymbol{z}$ has one component for each output class)

Probability distribution from scores

- Given the scores of an instance, we can get the probability distribution over all output classes.

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} P(Y = 1|\boldsymbol{x}, \boldsymbol{W}) \\ P(Y = 2|\boldsymbol{x}, \boldsymbol{W}) \\ \vdots \\ P(Y = k|\boldsymbol{x}, \boldsymbol{W}) \end{bmatrix}$$

- Use exp to rescale the scores, and normalize.

$$\boldsymbol{y} = \begin{bmatrix} \frac{\exp(z_1)}{\sum_i \exp(z_i)} \\ \frac{\exp(z_2)}{\sum_i \exp(z_i)} \\ \vdots \\ \frac{\exp(z_k)}{\sum_i \exp(z_i)} \end{bmatrix}$$

Regularization (avoiding overfitting)

- The classifier can model the training data too closely.
- E.g. giving always probabilities of ~ 0 or ~ 1 to match the training labels.
- $\Rightarrow$ the classifier will then fail to generalize on unseen data.
- Which scores correspond to those extreme probabilities?
- Which weights yield such scores?

Regularization (avoiding overfitting)

- The classifier can model the training data too closely.
- E.g. giving always probabilities of ~ 0 or ~ 1 to match the training labels.
- $\Rightarrow$ the classifier will then fail to generalize on unseen data.
- Which scores correspond to those extreme probabilities?
  Positive or negative scores with large absolute value.
- Which weights yield such scores?
  Positive or negative weights with large absolute value.
- Solution: Penalize large values in $\boldsymbol{W}$
  - $L1$-regularization: add sum of absolute values of elements in $\boldsymbol{W}$ to optimization objective.
    $\Rightarrow$ Results in sparse feature matrices (most entries are 0)
  - $L2$-regularization: add sum of squares of elements in $\boldsymbol{W}$ to optimization objective.
    $\Rightarrow$ Results in dense feature matrices (most entries are $\neq 0$)

Remarks on learning the optimal weights

- Evaluate what probability the model would give the correct labels in the training data.
- Find values of $\boldsymbol{W}$ to optimize that probability
- MaxEnt is a convex problem, but does not have a closed-form solution.
- The field of *convex optimization* offers a range of algorithms to solve such problems. Typical algorithms iteratively compute and use the gradient (=derivatives of likelihood with respect to weights).

Using the MaxEnt Model

MaxEnt Model in Scikit-Learn

- Hyper-parameters for learning:
  - `penalty`: `l1` or `l2`
  - C: inverse of regularization strength
  - `solver`, ...
  - etc
- Input for learning:
  $\boldsymbol{X}$: design matrix (shape: num_instances $\times$ features)
  $\boldsymbol{y}$: label (vector with num_instances integer elements, indicating the correct class 0,1,...)
- To analyse features you can look at the weights

```
from sklearn import linear_model
#(X, y) = (features matrix, labels)
maxent = linear_model.LogisticRegression(penalty='l2', C=1.0))
maxent.fit(X_train, y_train)
print maxent.coef_
# That is a matrix with the shape (n_classes, n_features)
```

Prediction and Evaluation

```
from sklearn.metrics import accuracy_score
# Predicts vector with (integer) labels.
y_predicted = maxent.predict(X_dev)
dev_acc = accuracy_score(y_dev, y_predicted)

# Probability distribution over all possible classes
# Shape: (n_instances, n_classes)
y_probs = maxent.predict_proba(X_dev)
```

... computing the F-scores for all classes:

```
>>> from sklearn.metrics import f1_score
>>> y_true = [0, 1, 2, 0, 1, 2]
>>> y_pred = [0, 2, 1, 0, 0, 1]
>>> f1_score(y_true, y_pred, average=None)
array([ 0.8,  0. ,  0. ])
```

Using Scikit-learn Maxent for paraphrase detection

- How to get X and y?
- Hyper-parameters to optimize?

Summary: Maximum Entropy model

- One of the most common classification models
- linear regression + scaling + normalization
- Can be interpreted as probabilities
- Easy to use in Sciki-learn