# Project 1 Tutorial: SVM Classification

EECS 445 WN2020

# Note

- This tutorial presentation is only an introduction to the project. You should still read the project  specs for details on the write up (and for a complete, comprehensive description of the project).
- Everything on the report that is highlighted in yellow are things that are to be reported in your project report
- For more on Python and scikit learn packages:
  - Consult jupyter notebook tutorials
  - Consult online scikit and python resources
- For more on SVM's
  - Consult lecture, lecture notes, and discussion notes

# Content


1. Project Introduction
   a. Problem and Dataset Introduction
   b. Learning Goals
   c. Python Requirements
2. Data Preprocessing
3. SVM Models and Hyperparameter Selection
4. Class Imbalances
5. Challenge
6. Demo

## Problem Intro

> This teacher is impressively dull. If you wanted compare this course to a food, it would be a cross between tofu and spam. The only excitement you do get is the bizarre commentary in response to your answers in class.

# Learning Goals

**Main Goal**: Learn how to carry out an applied ML project

Other learning goals: Learn about

- Data Processing and Feature Selection
- SVM Model and Hyperparameter Selection
- Performance Measure
- Class Imbalances
- Multiclass Classification

# 1. Python Requirements

- Python 3.6 (Python 3.7 is fine if no problems arise)
- Scikit-learn v0.20.2
- Numpy v1.15
- Pandas v0.24.0
- Matplotlib v3.0.2

# 2. Data Preprocessing and Feature Selection

- One key issues in NLP is how to obtain features from text data
- The method we will use is a “bag of words” model
  - There is a column/feature for each word
  - Two possible values
    - 1 if word is in text (regardless of number of occurrences)
    - 0 otherwise

# 2. Data Preprocessing and Feature Selection: Example

The cat is good

The cat is not good

The cat is not not good

| the | cat | is | not | good |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

# 2. Data Preprocessing and Feature Selection: Functions

- **extract_dictionary** (2.a)
  - Input: matrix X = [$\bar{x}^{(1)}$, $\bar{x}^{(2)}$, ..., $\bar{x}^{(n)}$ ], where $\bar{x}^{(i)}$= “i-th review”
  - Output: “dictionary” of words
  - Idea: go through all words in all reviews, and add them to the dictionary if they are not added yet

# 2. Data Preprocessing and Feature Selection: Functions

```python
def extract_dictionary(df):
    """
    Reads a panda dataframe, and returns a dictionary of distinct words
    mapping from each distinct word to its index (ordered by when it was found).
    Input:
        df: dataframe/output of load_data()
    Returns:
        a dictionary of distinct words that maps each distinct word
        to a unique index corresponding to when it was first found while
        iterating over all words in each review in the dataframe df
    """
    word_dict = {}
    # TODO: Implement this function
    return word_dict
```

# 2. Data Preprocessing and Feature Selection: Functions

- **generate_feature_matrix** (2.b)
  - Input: matrix X, dictionary of words
  - Output: feature_matrix
  - Idea: make a new matrix where each datapoint is now a bag-of-words vector

# 2. Data Preprocessing and Feature Selection: Functions

```python
def generate_feature_matrix(df, word_dict):
    """
    Reads a dataframe and the dictionary of unique words
    to generate a matrix of {1, 0} feature vectors for each review.
    Use the word_dict to find the correct index to set to 1 for each place
    in the feature vector. The resulting feature matrix should be of
    dimension (number of reviews, number of words).
    Input:
        df: dataframe that has the ratings and labels
        word_list: dictionary of words mapping to indices
    Returns:
        a feature matrix of dimension (number of reviews, number of words)
    """
    number_of_reviews = df.shape[0]
    number_of_words = len(word_dict)
    feature_matrix = np.zeros((number_of_reviews, number_of_words))
    # TODO: Implement this function
    return feature_matrix
```

# 3. Hyperparameter and Models

- In this part of the project, we will:
  - Explore different SVM’s by changing
    - Regularization function
    - Regularization hyperparameter
    - Kernel
  - Explore how regularization affects sparsity
  - Note: you also need to explore different performance metrics, but this will be covered in depth when we talk about class imbalances

# 3. Hyperparameter and Models: SVM Formulation

$$\underset{\bar{\theta}, b, \xi_i}{\text{minimize}} \frac{||\bar{\theta}||^2}{2} + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to } y^{(i)}(\bar{\theta} \cdot \bar{x}^{(i)} + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, \forall i = 1, 2, ..., n$$

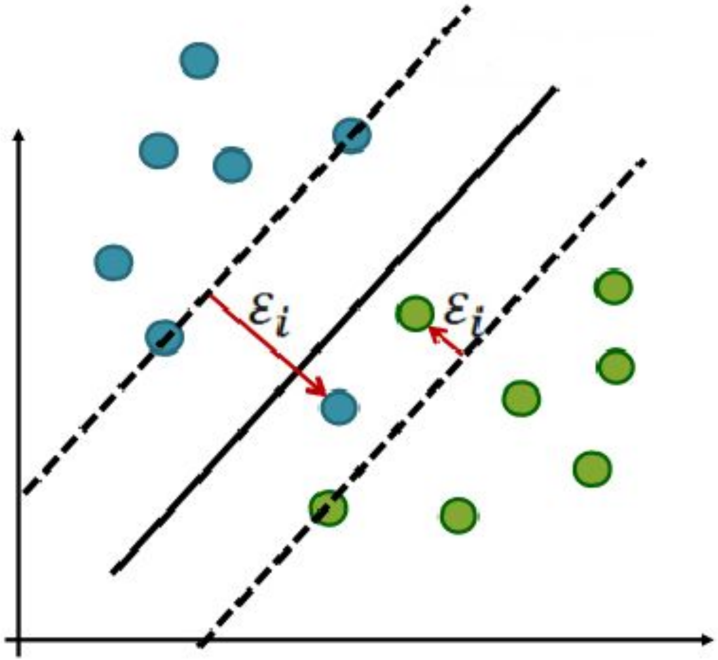
You do not need to implement these; you just need to use the SVM library

# 3. Hyperparameter and Models: SVM Formulation

$$\underset{\bar{\theta},b,\xi_i}{\text{minimize}} \frac{||\bar{\theta}||^2}{2} + C\sum_{i=1}^{n}\xi_i$$

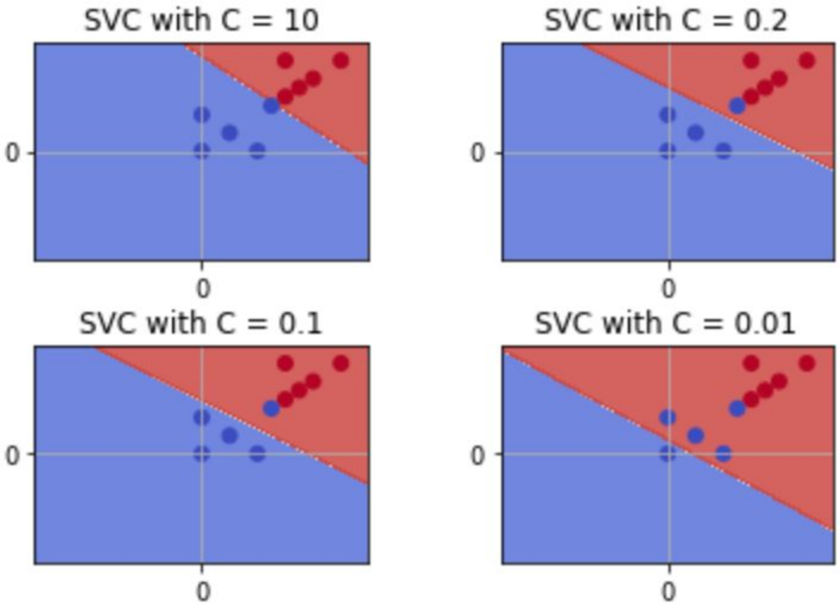
$$\text{subject to } y^{(i)}(\bar{\theta}\cdot\bar{x}^{(i)} + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, \forall i = 1, 2, ..., n$$



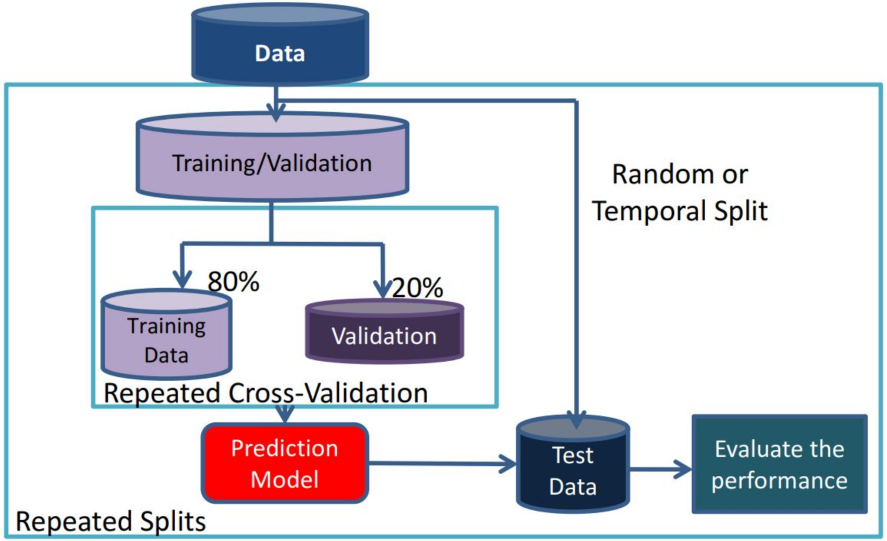
# 3. Hyperparameter and Models: SVM Formulation

- The hyperparameter C controls the balance between penalizing misclassifications and minimizing theta
- In the extreme, as C tends to infinity it becomes the hard margin SVM.



# 3. Hyperparameter and Models: Cross-Validation

- Cross validation is a technique used to ensure that we are not overfitting our training data
- We use cross-validation to find the "best" hyperparameters for our model



# 3. Hyperparameter and Models: Functions

- **cv_performance** (3.a)
    - Input: classifier, dataset, k, metric
    - Output: cross-validation performance
    - Given a dataset and a classifier, we will perform k-fold cross validation to maximize the metric of choice
    - *StratifiedKFold* class and its class method *split()* will come in handy

# 3. Hyperparameter and Models: Functions

```python
def cv_performance(clf, X, y, k=5, metric="accuracy"):
    """
    Splits the data X and the labels y into k-folds and runs k-fold
    cross-validation: for each fold i in 1...k, trains a classifier on
    all the data except the ith fold, and tests on the ith fold.
    Calculates the k-fold cross-validation performance metric for classifier
    clf by averaging the performance across folds.
    Input:
        clf: an instance of SVC()
        X: (n,d) array of feature vectors, where n is the number of examples
           and d is the number of features
        y: (n,) array of binary labels {1,-1}
        k: an int specifying the number of folds (default=5)
        metric: string specifying the performance metric (default='accuracy'
             other options: 'f1-score', 'auroc', 'precision', 'sensitivity',
             and 'specificity')
    Returns:
        average 'test' performance across the k folds as np.float64
    """
    # TODO: Implement this function
    #HINT: You may find the StratifiedKFold from sklearn.model_selection
    #to be useful

    #Put the performance of the model on each fold in the scores array
    scores = []

    #And return the average performance across all fold splits.
    return np.array(scores).mean()
```

# 3. Hyperparameter and Models: Functions

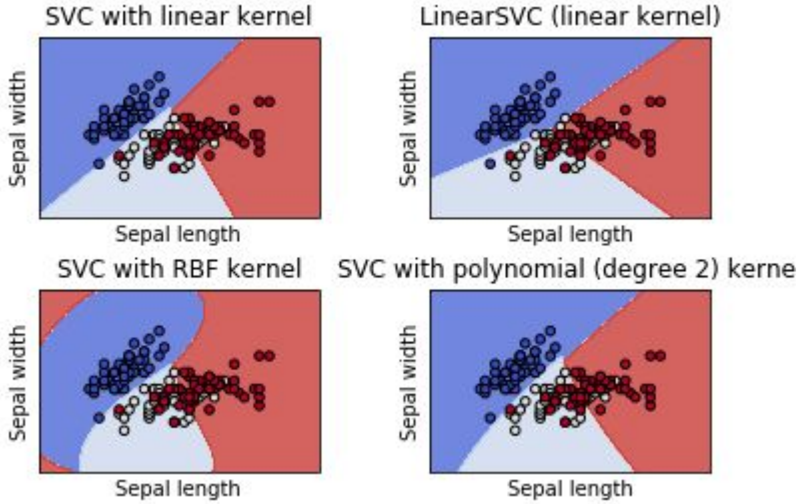
- **select_param_linear** (3.b)
  - Input: dataset, k, metric, C range, penalty
  - Output: optimal C value
  - We will obtain the best C value (measured by metric) for a linear kernel SVM by performing k-fold cross validation on the dataset for each C
  - Will call cv_performance

# 3. Hyperparameter and Models: Functions

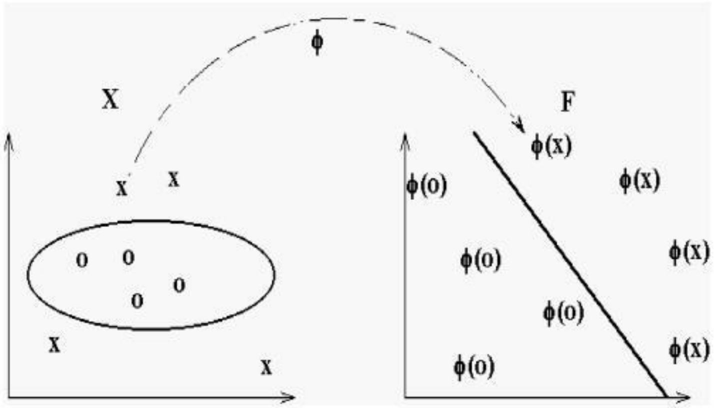
```python
def select_param_linear(X, y, k=5, metric="accuracy", C_range = [], penalty='l2'):
    """
    Sweeps different settings for the hyperparameter of a linear-kernel SVM,
    calculating the k-fold CV performance for each setting on X, y.
    Input:
        X: (n,d) array of feature vectors, where n is the number of examples
        and d is the number of features
        y: (n,) array of binary labels {1,-1}
        k: int specifying the number of folds (default=5)
        metric: string specifying the performance metric (default='accuracy',
             other options: 'f1-score', 'auroc', 'precision', 'sensitivity',
             and 'specificity')
        C_range: an array with C values to be searched over
    Returns:
        The parameter value for a linear-kernel SVM that maximizes the
        average 5-fold CV performance.
    """
    # TODO: Implement this function
    #HINT: You should be using your cv_performance function here
    #to evaluate the performance of each SVM
    return 0.0
```

# 3. Hyperparameter and Models: Kernels

- Another advantage of SVM’s is that we can efficiently have non-linear classifiers by choosing different kernel functions/ feature mappings
- Common Kernels:
  - Linear
  - Polynomial
  - rbf



# 3. Hyperparameter and Models: Quadratic Kernel




$$K(\bar{x}, \bar{x}') = \phi(\bar{x}) \cdot \phi(\bar{x}')$$

$$K(\bar{x}, \bar{x}') = (\bar{x} \cdot \bar{x}' + r)^2$$

Hyperparameter r controls scaling of the linear terms

# 3. Hyperparameter and Models: Functions

- **select_param_quadratic** (3.b)
    - Input: dataset, k, metric, (C, r) values, penalty
    - Output: optimal (C, r) value
    - We will obtain the best (C, r) value (measured by metric) for a quadratic kernel SVM by performing k-fold cross validation on the dataset for each (C, r)
    - Will call cv_performance

# 3. Hyperparameter and Models: Functions

```python
def select_param_quadratic(X, y, k=5, metric="accuracy", param_range=[]):
    """
        Sweeps different settings for the hyperparameters of an quadratic-kernel SVM,
        calculating the k-fold CV performance for each setting on X, y.
        Input:
            X: (n,d) array of feature vectors, where n is the number of examples
               and d is the number of features
            y: (n,) array of binary labels {1,-1}
            k: an int specifying the number of folds (default=5)
            metric: string specifying the performance metric (default='accuracy'
                     other options: 'f1-score', 'auroc', 'precision', 'sensitivity',
                     and 'specificity')
            parameter_values: a (num_param, 2)-sized array containing the
                parameter values to search over. The first column should
                represent the values for C, and the second column should
                represent the values for r. Each row of this array thus
                represents a pair of parameters to be tried together.
        Returns:
            The parameter value(s) for a quadratic-kernel SVM that maximize
            the average 5-fold CV performance
    """
    # TODO: Implement this function
    # Hint: This will be very similar to select_param_linear, except
    # the type of SVM model you are using will be different...
```

# 3. Hyperparameter and Models: Regularization and Sparsity

- The choice of regularization loss and hyperparameter directly affects how sparse a solution is (more spare -> more elements are equal to 0 in theta)
- We will measure sparsity of our solution by using the L0 norm

$$||\bar{\theta}||_0 = \sum_{i=1}^{d} \mathbb{I}\{\theta_i \neq 0\}$$

# 3. Hyperparameter and Models: Regularization and Sparsity

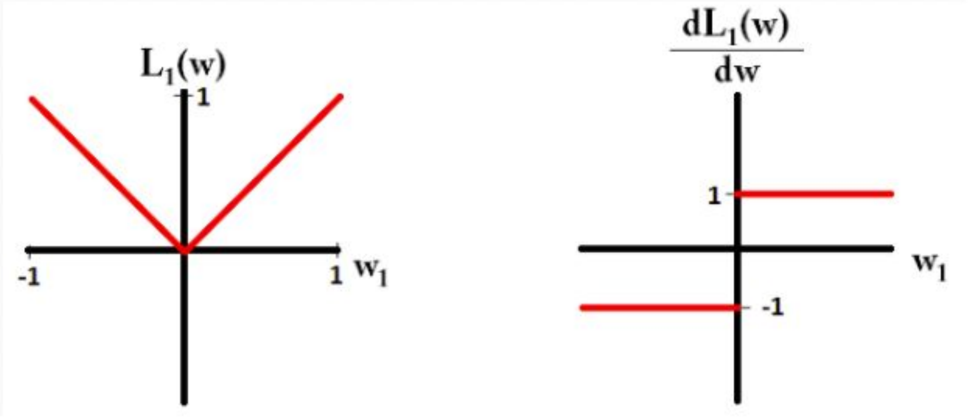



Figure 2: The $\ell_1$-norm and its gradient

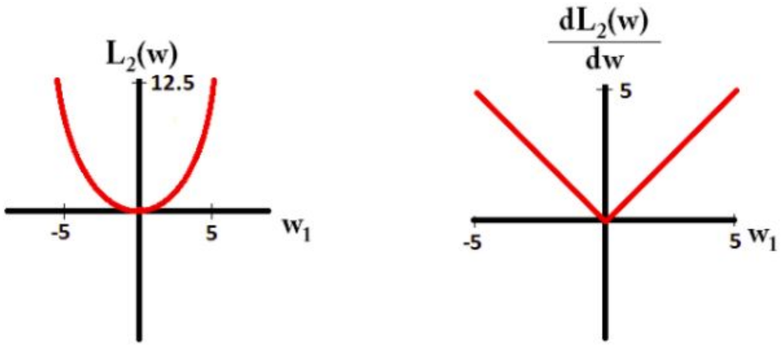



Figure 3: The $\ell_2$-norm and its gradient

# 3. Hyperparameter and Models: Functions

- plot_weight
    - Input: dataset, penalty, metric, C range
    - Output: plot the L0 norm depending on each C value

# 3. Hyperparameter and Models: Functions

```python
def plot_weight(X,y,penalty,metric,C_range):
    """
    Takes as input the training data X and labels y and plots the L0-norm
    (number of nonzero elements) of the coefficients learned by a classifier
    as a function of the C-values of the classifier.
    """

    print("Plotting the number of nonzero entries of the parameter vector as a function of C")
    norm0 = []

    # TODO: Implement this part of the function
    #Here, for each value of c in C_range, you should
    #append to norm0 the L0-norm of the theta vector that is learned
    #when fitting an L2-penalty, degree=1 SVM to the data (X, y)
```

# 4. Class Imbalances

What happens when we have many more negative examples than positive examples?

- 40% positive 60% negative?
- 90% positive 10% negative?

# 4. Class Imbalances: Evaluation

- Accuracy
- Precision
- Sensitivity
- Specificity
- F1-Score
- AUROC

# 4. Class Imbalances: Evaluation

| | | Actual label | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted label | Positive | TP | FP |
| | Negative | FN | TN |

# 4. Class Imbalances: Accuracy

| | | Actual label | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted label | Positive | TP | FP |
| | Negative | FN | TN |

$$\frac{TP + TN}{TP + TN + FP + FN}$$

# 4. Class Imbalances: Precision

| | | Actual label | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted label | Positive | TP | FP |
| | Negative | FN | TN |

$$\frac{TP}{TP + FP}$$

# 4. Class Imbalances: Sensitivity

| | | Actual label | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted label | Positive | TP | FP |
| | Negative | FN | TN |

$$\frac{TP}{TP + FN}$$

# 4. Class Imbalances: Specificity

| | | Actual label | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted label | Positive | TP | FP |
| | Negative | FN | TN |

$$\frac{TN}{TN + FP}$$

# 4. Class Imbalances: F1-Score

$$\left(\frac{precision^{-1} + sensitivity^{-1}}{2}\right)^{-1} = \frac{2TP}{2TP + FP + FN}$$

# 4. Class Imbalances: AUROC

- Area Under Receiving Operating Characteristic (AUROC) Curve
- Measures the trade-off between true positive rate and false positive rate (ranges between 0 and 1)
- Measures the probability that a randomly selected positive point is ranked higher than a randomly selected negative point

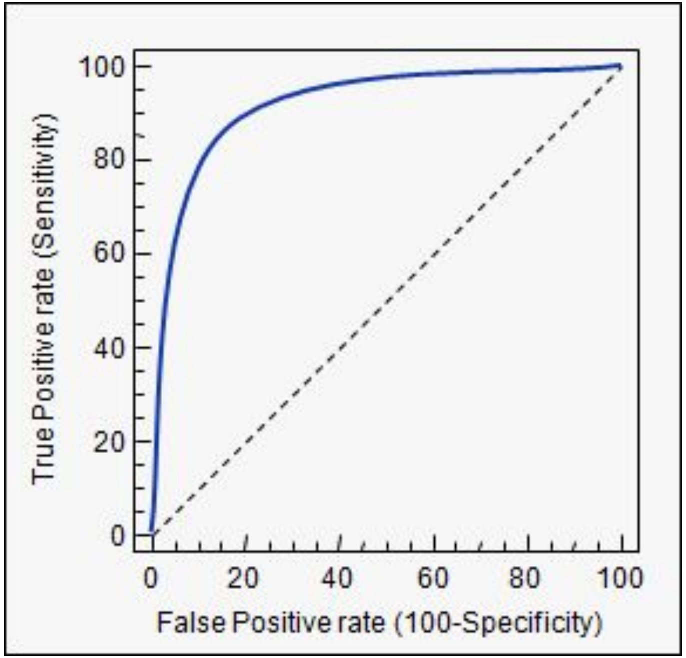
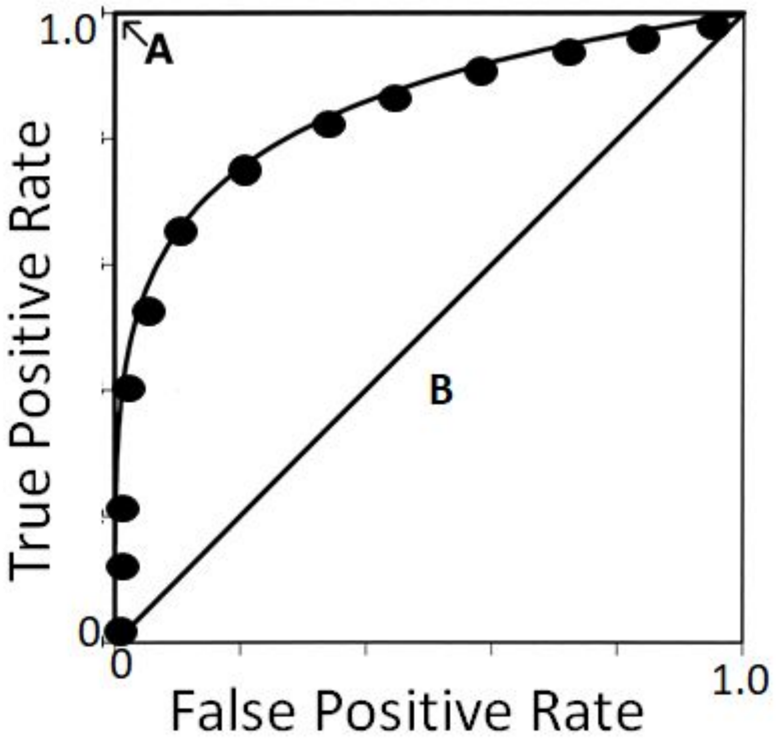



Image Credit: https://www.medcalc.org/manual/roc-curves.php

# 4. Class Imbalances: AUROC

- Each point corresponds with a single decision boundary
- Decision boundaries created by adjusting the threshold for prediction



# 4. Class Imbalances: Evaluation

```python
def performance(y_true, y_pred, metric="accuracy"):
    """
    Calculates the performance metric as evaluated on the true labels
    y_true versus the predicted labels y_pred.
    Input:
        y_true: (n,) array containing known labels
        y_pred: (n,) array containing predicted scores
        metric: string specifying the performance metric (default='accuracy'
                other options: 'f1-score', 'auroc', 'precision', 'sensitivity',
                and 'specificity')
    Returns:
        the performance as an np.float64
    """
    # TODO: Implement this function
    # This is an optional but very useful function to implement.
    # See the sklearn.metrics documentation for pointers on how to implement
    # the requested metrics.
```

# 4. Class Imbalances: Class Weights

Assign weights to each class in the cost function

$$\underset{\bar{\theta},b,\xi_i}{\text{minimize}} \frac{||\bar{\theta}||^2}{2} + W_p * C \sum_{i|y^{(i)}=1} \xi_i + W_n * C \sum_{i|y^{(i)}=-1} \xi_i$$

$$\text{subject to } y^{(i)}(\bar{\theta} \cdot \phi(\bar{x}^{(i)}) + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, \forall i = 1, 2, 3, ..., n$$

# 4. Class Imbalances: Class Weights

```python
def select_classifier(penalty='l2', c=1.0, degree=1, r=0.0, class_weight='balanced'):
    """
    Return a linear svm classifier based on the given
    penalty function and regularization parameter c.
    """
    # TODO: Optionally implement this helper function if you would like to
    # instantiate your SVM classifiers in a single function. You will need
    # to use the above parameters throughout the assignment.
```

# 5. Challenge

- In the challenge portion, we encourage you to explore the tools you have learned so far to find the best classifier
  - Train a classifier and make predictions for held-out data
  - Explore different features, SVM’s, kernels, loss functions, etc.
  - Note: you are not required to explore all techniques, but you are encouraged to do so
- Using any amount of training data you want (time / accuracy tradeoff)
- 50% of the grade is effort; 50% is performance (normalized by how well the class performs)

# 5. Challenge: Problem Intro

Given the text of a RMP review and additional features, can we determine the sentiment of the RMP review?

$\mathbf{x}^{(i)}$ = [ ReviewText, ReviewTime, unixReviewTime, Summary]

$y^{(i)}$ = {-1, 0, +1}

- Multiclass classification
    - One vs. One and One vs. Rest

# 5. Challenge: Feature Engineering

- Using a different method for extracting words from the review
- Using only a subset of the raw features
- Utilizing more info such as rating, summary, etc.
- Scaling or normalizing the data
- Alternative feature representations
  - current: One-hot encoding.
  - Other encoding methods?

# 5. Challenge

- Time variables
  - The pandas library offers nice, easy ways of parsing date data into a numerical value
  - Often it is useful to extract more features from that date numerical value to account for cyclical nature or for trends
- Missing values
  - Real data is messy, and often we get data points that do not have all of the features